Theoretical Computer Science 411 (2010) 1283–1310

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Basing cryptographic protocols on tamper-evident seals

Tal Moran *, Moni Naor [1]

Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel

ARTICLE INFO

Keywords:
Tamper-evident seals
Human
Protocol
Universally composable
Coin flipping

ABSTRACT

In this article, we attempt to formally study two very intuitive physical models: sealed envelopes and locked boxes, often used as illustrations for common cryptographic operations. We relax the security properties usually required from locked boxes [such as in bit-commitment (BC) protocols] and require only that a broken lock or torn envelope be identifiable to the original sender. Unlike the completely impregnable locked box, this functionality may be achievable in real life, where containers having this property are called "tamper-evident seals". Another physical object with this property is the "scratch-off card", often used in lottery tickets. We consider three variations of tamper-evident seals, and show that under some conditions they can be used to implement oblivious transfer, BC and coin flipping (CF). We also show a separation between the three models. One of our results is a strongly fair CF protocol with bias bounded by $O(1/r)$ (where r is the number of rounds); this was a stepping stone towards achieving such a protocol in the standard model (in subsequent work).

1. Introduction

In this article, we consider the use of "tamper-evident seals" in cryptographic protocols. A tamper-evident seal is a primitive based on very intuitive physical models: the sealed envelope and the locked box. In the cryptographic and popular literature, these are often used as illustrations for a number of basic cryptographic primitives. For instance, when Alice sends an encrypted message to Bob, she is often depicted as placing the message in a locked box and sending the box to Bob (who needs the key to read the message).

Bit commitment (BC), another well known primitive, is usually illustrated using a sealed envelope. In a BC protocol one party, Alice, commits to a bit b to Bob in such a way that Bob cannot tell what b is. At a later time, Alice can reveal b, and Bob can verify that this is indeed the bit to which she committed. The standard illustration used for a BC protocol is Alice putting b in a sealed envelope, which she gives to Bob. Bob cannot see through the envelope (so cannot learn b). When Alice reveals her bit, she lets Bob open the envelope so he can verify that she did not cheat.

The problem with the above illustration is that a physical "sealed envelope", used in the simple manner described, is insufficient for BC: Bob can always tear open the envelope before Alice officially allows him to do so. Even a locked box is unlikely to suffice; many protocols based on BC remain secure only if no adversary can *ever* open the box without a key. A more modest security guarantee seems to be more easily obtained: an adversary may be able to tear open the envelope but Alice will be able to recognize this when she sees the envelope again.

"Real" closures with this property are commonly known as "tamper- evident seals". These are used widely from containers for food and medicines to high-security government applications. Another common application that embodies

* Present address: Center for Research on Computation and Society, Harvard University, Cambridge, MA 02138, USA.
E-mail addresses: talm@seas.harvard.edu (T. Moran), moni.naor@weizmann.ac.il (M. Naor).

[1] Incumbent of the Judith Kleeman Professorial Chair.

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.10.023

these properties is the "scratch-off card", often used as a lottery ticket. This is usually a printed cardboard card which has some areas coated by an opaque layer (e.g., the possible prizes to be won are covered). The text under the opaque coating cannot be read without scratching off the coating, but it is immediately evident that this has been done (so the card issuer can verify that only one possible prize has been uncovered).

In this article, we attempt to clarify what it means to use a sealed envelope or locked box in a cryptographic protocol. Our focus is on constructing cryptographic protocols that use physical tamper-evident seals as their basis. In particular, we study their applicability to coin flipping (CF), zero-knowledge protocols, BC and oblivious transfer (OT), some of the most fundamental primitives in modern cryptography; OT is sufficient by itself for secure function evaluation [16,18] without additional complexity assumptions. OT implies BC, which in turn implies zero-knowledge proofs for any language in NP [15] and (weakly fair) CF [6].

Note that encryption is very simple to implement using tamper-evident containers (given authenticated channels), which is why we do not discuss in depth in this article. For example, Alice and Bob can agree on a secret key by sending random bits in sealed containers. A bit in a container that arrives unopened is guaranteed (by the tamper-evidence property) to be completely unknown to the adversary. The case where only the creator of a container can tell whether it has been opened requires only slightly more complex protocols.

1.1. Seals in different flavours

The intuitive definition of a tamper-evident seal does not specify its properties precisely. In this article, we consider three variants of containers with tamper-evident seals. The differences arise from two properties: whether or not sealed containers can be told apart and whether or not an honest player can break the seal.

Distinguishable vs. indistinguishable. One possibility is that containers can always be uniquely identified, even when sealed (e.g., the containers have a serial number engraved on the outside). We call this a "distinguishable" model. A second possibility is that containers can be distinguished only when open; all closed containers look alike, no matter who sealed them (this is similar to the paper-envelope voting model, where the sealed envelopes cannot be told apart). We call this an "indistinguishable" model.

Weak lock vs. envelope. The second property can be likened to the difference between an envelope and a locked box: an envelope is easy to open for anyone. A locked box, on the other hand, may be difficult for an "honest" player to open without a key, although a dishonest player may know how to break the lock. We call the former an "envelope" model and the latter a "weak lock" model. In Section 2, we give formal definitions for the different models.

Any envelope model is clearly stronger than the corresponding weak lock model (since in the envelope model the honest player is more powerful, while the adversary remains the same). We show that there are protocols that can be implemented in the indistinguishable models that cannot be realized in any of the distinguishable models. It is not clear, however, that any indistinguishable model is strictly stronger than any distinguishable model. Although all four combinations are possible, the indistinguishable envelope model does not appear significantly stronger than the indistinguishable weak lock (IDWL) model, and in this article we discuss only the latter. Note that in the standard model of cryptography, where the parties exchange messages and there is no access to outside physical resources, we do not know how to implement any of these closures.

Additional variants. The definitions of tamper-evident seals we consider in this article are by no means the only possible ones. They do, however, represent a fairly weak set of requirements for a physical implementation. In particular, we do not require the containers to be unforgeable by their creator (this relaxation is captured by allowing the creator of the container to change its contents and reseal it).

1.2. Our results

In this article, we show that tamper-evident seals can be used to implement standard cryptographic protocols. We construct protocols for "weakly fair" CF (in which the result is 0, 1 or *invalid*), BC and OT using tamper-evident seals as primitives.

A possibly practical application of our model is the "cryptographic randomized response technique" (CRRT), defined by Ambainis et al. [2]. "Randomized response" is a polling technique used when some of the answers to the poll may be stigmatizing (e.g., "do you use drugs?"). The respondent lies with some known probability, allowing statistical analysis of the results while letting the respondent disavow a stigmatizing response. In a CRRT, there is an additional requirement that a malicious respondent cannot bias the results more than by choosing a different answer. The techniques described by Ambainis et al. achieve this, but require "heavy" cryptographic machinery (such as OT), or quantum cryptography. In a follow-up article [21], we show a simple protocol for CRRT using scratch-off cards.

One of the most interesting results is a protocol for "strongly fair" CF (where the result for an honest player must be either 0 or 1 even if the other player quits before finishing the protocol) with bias bounded by $O(\frac{1}{r})$, where r is the number of rounds. This protocol was a stepping-stone to the subsequent construction of an optimal protocol for strongly fair CF in the standard model [22].

Table 1
Comparison of tamper-evident seal models.

Model	Possible	Impossible
Bare		CF, BC, OT
Dist. weak locks	CF	BC, OT
Dist. envelopes	CF, BC, Strongly fair CF$(1/r)$	OT
Indist. weak locks	CF, BC, OT	??

An important contribution of this article is the *formal* analysis for the models and protocols we construct. We show that the protocols are *Universally Composable* (UC) in the sense of Canetti [7]. This allows us to use them securely as "black-boxes" in larger constructions.

On the negative side, we show that our protocol for strongly fair CF using sealed envelopes is optimal: it is impossible to do better than $O(\frac{1}{r})$ bias (this follows from a careful reading of the proof in [8]). We also give impossibility results for BC and OT (note that we show the impossibility of *any* type of BC or OT, not just UC realizations). The proofs are based on information-theoretic methods: loosely speaking, we show that the sender has too much information about what the receiver knows. When this is the case, BC is impossible because the sender can decide in advance what the receiver will accept (so either the receiver knows the committed bit or it is possible to equivocate), while OT is impossible because the transfer cannot be "oblivious" (the sender knows how much information the receiver has on each of his bits).

Our results show a separation between the different models of tamper-evident seals and the "bare" model, summarized in Table 1:

1.3. Related work

To the best of our knowledge, this is the first attempt at using tamper-evident seals for cryptographic protocols. Ross Anderson discusses "packaging and seals" in the context of security engineering [3], however the use of tamper-evidence does not extend to more complex protocols. Blaze gives some examples of the reverse side of the problem: cryptanalysis of physical security systems using techniques from computer science [4,5]. Using scratch-off cards in the lottery setting can be described as a very weak form of CF, however, we do not believe this has ever been formally analyzed (or used in more complex protocols).

On the other hand, basing cryptographic protocols on physical models is a common practice. Perhaps, the most striking example is the field of quantum cryptography, where the physics of quantum mechanics are used to implement cryptographic operations — some of which are impossible in the "bare" model. One of the inspirations for this work was the idea of "Quantum Bit Escrow" (QBE) [1], a primitive that is very similar to a tamper-evident seal and that can be implemented in a quantum setting. There are, however, significant differences between our definitions of tamper-evident seals and QBE. In particular, in QBE the adversary may "entangle" separate escrowed bits and "partially open" commitments. Thus, while unconditionally secure BC is impossible in the pure quantum setting [20,19], it is possible in ours.

Much work has been done on basing BC and OT on the physical properties of communication channels, using the random noise in a communication channel as the basis for security. Both BC and OT were shown to be realizable in the *Binary Symmetric Channel* model [10,9], in which random noise is added to the channel in both directions with some known, constant, probability. Later work shows that they can also be implemented, under certain conditions, in the weaker (but more convincing) *Unfair Noisy Channel* model [12,11], where the error probability is not known exactly to the honest parties, and furthermore can be influenced by the adversary. Our construction for 1–2 OT uses some of the techniques and results from [12].

One of the motivations for this work was the attempt to construct cryptographic protocols that are implementable by humans without the aid of computers. This property is useful, for example, in situations where computers cannot be trusted to be running the protocol they claim, or where "transparency" to humans is a requirement (such as in voting protocols). Many other examples exist of using simple physical objects as a basis for cryptographic protocols that can be performed by humans, some are even folklore: Sarah Flannery [14] recounts a childhood riddle that uses a doubly locked box to transfer a diamond between two parties, overcoming the corrupt postal system (which opens any unlocked boxes) despite the fact that the two parties have never met (and can only communicate through the mail). Fagin, Naor and Winkler [13] assembled a number of solutions to the problem of comparing secret information without revealing anything but the result of the comparison using a variety of different physical methods. Schneier devised a cipher [24] that can be implemented by a human using a pack of cards. In a lighter vein, Naor, Naor and Reingold [23] give a protocol that provides a "zero-knowledge proof of knowledge" of the correct answer to the children's puzzle "Where's Waldo" using only a large newspaper and scissors. A common thread in these works is that they lack a formal specification of the model they use, and a formal proof of security.

1.4. Organization of the article

In Section 2, we give formal definitions for the different models of tamper-evident seals and the functionalities we attempt to realize using them. In Section 3, we discuss the capabilities of the distinguishable weak lock (DWL) model, show

that BC is impossible in this model and give a protocol for weakly fair CF. In Section 4, we discuss the capabilities of the distinguishable envelope (DE) model, showing that OT is impossible and giving protocols for BC and strongly fair CF with bias $1/r$. Section 5 contains a discussion of the indistinguishable weak lock model and a protocol for OT in this model. The proofs of security for the protocols we describe are given in Sections 6, 8.1, 7, 9 and 10. The proofs are fairly technical, and can be skipped on first reading. Section 11 contains the discussion and some open problems.

2. Model: Ideal functionalities

2.1. Ideal functionalities and the UC framework

Many two-party functionalities are easy to implement using a trusted third party that follows pre-agreed rules. In proving that a two-party protocol is secure, we often want to say that it behaves "as if it were performed using the trusted third party". A formalization of this idea is the UC model defined by Canetti [7].

In the UC model, the trusted third party is called the *ideal functionality*. The ideal functionality is described by a program (formally, it is an interactive Turing machine) that can communicate by authenticated, private channels with the participants of the protocol.

The notion of security in the UC model is based on simulation: a protocol securely realizes an ideal functionality in the UC model if any attack on the protocol in the "real" world, where no trusted third party exists, can be performed against the ideal functionality with the same results. Attacks in the ideal world are carried out by an "ideal adversary", that can also communicate privately with the functionality. The ideal adversary can corrupt honest parties by sending a special **Corrupt** command to the functionality, at which point the adversary assumes full control of the corrupted party. This allows the functionality to act differently depending on which of the parties are corrupted. Additional capabilities of the adversary are explicitly defined by the ideal functionality.

Proving protocol security in the UC model provides two main benefits: First, the functionality definition is an intuitive way to describe the desired properties of a protocol. Second (and the original motivation for the definition of the UC model), protocols that are secure in the UC have very strong security properties, such as security under composition and security that is retained when the protocol is used as a sub-protocol to replace an ideal functionality. This security guarantee allows us to simplify many of our proofs, by showing separately the security of their component sub-protocols.

Note that our impossibility results are not specific to the UC model: the impossibility results for BC (Section 3.3), OT (Section 4.1) and the lower bound for strongly fair CF (Section 4.4) hold even for the weaker "standard" notions of these functionalities.

In this section, we formally define the different models for tamper-evident seals in terms of their ideal functionalities. For completeness, we also give the definitions of the primitives we are trying to implement (CF, BC and OT). We restrict ourselves to the two-party case, and to adversaries that decide at the beginning of the protocol whether to corrupt one of the parties or neither.

For readability, we make a few compromises in strict formality when describing the functionalities. First, the description is in natural language rather than pseudocode. Second, we implicitly assume the following for all the descriptions:

- All functionalities (unless explicitly specified) have a **Halt** command that can be given by the adversary at any time. When a functionality receives this command, it outputs $\perp$ to all parties. The functionality then halts (ignoring further commands). In a two party protocol, this is equivalent to a party halting prematurely.
- When a functionality receives an invalid command (one that does not exist or is improperly formatted), it proceeds as if it received the **Halt** command.
- When we say that the functionality "verifies" some condition, we mean that if the condition does not hold, the functionality proceeds as if it received the **Halt** command.

2.2. Tamper-evident seals

These are the functionalities on which we base the protocols we describe in the article. In the succeeding sections, we assume we are given one of these functionalities and attempt to construct a protocol for a "target" functionality (these are described in Section 2.3).

2.2.1. Distinguishable weak locks

This functionality models a tamper-evident container that has a "weak lock": an honest party requires a key to open the container, but the adversary can break the lock without help. Functionality $\mathcal{F}^{(DWL)}$ contains an internal table that consists of tuples of the form $(id, value, creator, holder, state)$. The table represents the state and location of the tamper-evident containers. It contains one entry for each existing container, indexed by the container's id and creator. We denote $value_{id}$, $creator_{id}$, $holder_{id}$ and $state_{id}$ the corresponding values in the table in row id (assuming the row exists). The table is initially empty. The functionality is described as follows, running with parties $P_1, \ldots, P_n$ and ideal adversary $\mathcal{I}$:

Seal (id, $value$) This command creates and seals a container. On receiving this command from party P_i, the functionality verifies that id has the form $(P_i, \{0,1\}^*)$ (this form of id is a technical detail to ensure that ids are local to each party). If this is the first **Seal** message with id id, the functionality stores the tuple (id, $value$, P_i, P_i, **sealed**) in the table. If this is not the first **Seal** with id id, it verifies that $creator_{id} = holder_{id} = P_i$ and, if so, replaces the entry in the table with (id, $value$, P_i, P_i, **sealed**).

Send (id, P_j) On receiving this command from party P_i, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. If so, it outputs $(\textbf{Receipt}, id, creator_{id}, P_i, P_j)$ to P_j and $\mathcal{A}$ and replaces the entry in the table with $(id, value_{id}, creator_{id}, P_j, state_{id})$.

Open id On receiving this command from P_i, the functionality verifies that an entry for container id appears in the table, that $holder_{id} = P_i$ and that either P_i is corrupted or $state_{id} =$ **unlocked**. It then sends (**Opened**, id, $value_{id}$, $creator_{id}$) to P_i. If $state_{id} \neq$ **unlocked** it replaces the entry in the table with (id, $value_{id}$, $creator_{id}$, $holder_{id}$, **broken**).

Verify id On receiving this command from P_i, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. It then considers $state_{id}$. If $state_{id} =$ **broken** it sends (**Verified**, id, **broken**) to P_i. Otherwise, it sends (**Verified**, id, **ok**) to P_i.

Unlock id On receiving this command from P_i, the functionality verifies that an entry for container id appears in the table, that $creator_{id} = P_i$ and that $state_{id} =$ **sealed**. If so, it replaces the entry in the table with (id, $value_{id}$, $creator_{id}$, $holder_{id}$, **unlocked**) and sends (**Unlocked**, id) to $holder_{id}$.

2.2.2. Distinguishable envelopes

Functionality $\mathcal{F}^{(DE)}$ models a tamper-evident "envelope": in this case, honest parties can open the envelope without need for a key (although the opening will be evident to the envelope's creator if the envelope is returned). This functionality is almost exactly identical to $\mathcal{F}^{(DWL)}$, except the **Open** command allows anyone holding the container to open it. The functionality description is identical to $\mathcal{F}^{(DWL)}$, except that the new handling of the **Open** command is:

Open id On receiving this command from P_i, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. It sends (**Opened**, id, $value_{id}$, $creator_{id}$) to P_i. It also replaces the entry in the table with (id, $value_{id}$, $creator_{id}$, $holder_{id}$, **broken**).

The **Unlock** command is now irrelevant, but still supported to make it clear that this model is strictly stronger than the weak lock model.

2.2.3. Indistinguishable weak locks

This functionality models tamper-evident containers with "weak locks" that are indistinguishable from the outside. The indistinguishability is captured by allowing the players to shuffle the containers in their possession using an additional **Exchange** command. To capture the fact that the indistinguishability applies only to *sealed* containers, the internal table contains an addition column: sid, the "sealed id". This is a unique id that is shuffled along with the rest of the container contents and is revealed when the container is opened.[2]

Functionality $\mathcal{F}^{(IWL)}$ can be described as follows, running with parties $P_1, \ldots, P_n$ and adversary $\mathcal{A}$:

Seal (id, sid, $value$) This command creates and seals a container. On receiving this command from party P_i, the functionality verifies that id and sid have the form $(P_i, \{0,1\}^*)$.

Case 1: *This is the first **Seal** message with id id and sid sid.* In this case, the functionality stores the tuple (id, sid, $value$, P_i, P_i, **sealed**) in the table.

Case 2: *This is the first **Seal** message with sid sid but id has been used before.* In this case, the functionality verifies that $holder_{id} = P_i$. It then replaces the entry in the table with (id, sid, $value$, P_i, P_i, **sealed**).

Case 3: *This is the first **Seal** message with id id but sid has been used before.* In this case, the functionality proceeds as if it has received the **Halt** command.

Send (id, P_j) On receiving this command from party P_i, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. If so, it sends $(\textbf{Receipt}, id, P_i, P_j)$ to P_j and $\mathcal{A}$ and replaces the entry in the table with $(id, sid_{id}, value_{id}, creator_{id}, P_j, state_{id})$.

Open id On receiving this command from P_i, the functionality verifies that an entry for container id appears in the table, that $holder_{id} = P_i$ and that either P_i is corrupted or $state_{id} =$ **unlocked**. It then sends (**Opened**, id, sid_{id}, $value_{id}$, $creator_{id}$) to P_i. If $state_{id} \neq$ **unlocked** it replaces the entry in the table with (id, sid_{id}, $value_{id}$, $creator_{id}$, $owner_{id}$, **broken**).

Verify id On receiving this command from P_i, the functionality verifies that an entry for container id appears in the table and that $holder_{id} = P_i$. It then considers $state_{id}$. If $state_{id} =$ **broken** it sends (**Verified**, id, **broken**) to P_i. Otherwise, it sends (**Verified**, id, **ok**) to P_i.

[2] Technically, the sid can be used to encode more than a single bit in a container. We do not make use of this property in any of our protocols, but changing the definition to eliminate it would make it unduly cumbersome.

Unlock *sid* On receiving this command from P_i, the functionality verifies that an entry exists in the table for which $sid_{id} = sid$, that $creator_{id} = P_i$. If $state_{id} =$ **sealed**, it replaces the entry in the table with $(id, sid_{id}, value_{id}, creator_{id}, holder_{id},$ **unlocked**$)$. Otherwise, it does nothing. Note that this command does not send any messages (so it cannot be used to determine who is holding a container).

Exchange (id_1, id_2) On receiving this command from P_i the functionality verifies that both id_1 and id_2 exist in the table, and that $holder_{id_1} = holder_{id_2} = P_i$. It then exchanges the two table rows; the tuples in the table are replaced with $(id_2, sid_{id_1}, value_{id_1}, creator_{id_1}, P_i, state_{id_1})$ and $(id_1, sid_{id_2}, value_{id_2}, creator_{id_2}, P_i, state_{id_2})$.

A note about notation. In the interests of readability, we will often refer to parties "preparing" a container or envelope instead of specifying that they send a **Seal** message to the appropriate functionality. Likewise, we say a party "verifies that a container is sealed" when the party sends a **Verify** message to the functionality, waits for the response and checks that the resulting **Verified** message specifies an **ok** status. We say a party "opens a container" when it sends an **Open** message to the functionality and waits for the **Opened** response. We say the party "shuffles" a set of containers according to some permutation (in the indistinguishable model) when it uses the appropriate **Exchange** messages to apply the permutation to the containers' ids.

2.3. Target functionalities

These are the "standard" functionalities we attempt to implement using the tamper-evident seals.

2.3.1. Weakly fair coin flipping

This functionality models CF in which the result of the coin flip can be 0, 1 or $\perp$. The result of the flip c should satisfy: $\Pr[c = 0] \leq \frac{1}{2}$ and $\Pr[c = 1] \leq \frac{1}{2}$. This is usually what is meant when talking about "CF" (for instance, in Blum's "Coin Flipping Over the Telephone" protocol [6]). The $\perp$ result corresponds to the case where one of the parties noticeably deviated from (or prematurely aborted) the protocol. Under standard cryptographic assumptions (such as the existence of one-way functions), weakly fair CF is possible. Conversely, in the standard model the existence of weakly fair CF implies one-way functions [17].

Functionality $\mathcal{F}^{(WCF)}$ is described as follows, with parties Alice and Bob and adversary $\mathcal{A}$:

Value The sender of this command is Alice (the other party is Bob). When this command is received, the functionality chooses a uniform value $d \in \{0, 1\}$. If one of the parties is corrupted, the functionality outputs (**Approve**, d) to $\mathcal{A}$ (the adversary). In that case, the functionality ignores all inputs until it receives either a **Continue** command or a **Halt** command from $\mathcal{A}$. If no party is corrupted, the functionality proceeds as if $\mathcal{A}$ had sent a **Continue** command.

Halt When this command is received from $\mathcal{A}$ (in response to an **Approve** message) the functionality outputs $\perp$ to all parties and halts.

Continue When this command is received from $\mathcal{A}$ (in response to an **Approve** message), the functionality outputs (**Coin**, d) to all parties and halts.

Note: if only one of the parties can cheat in the coin flip, we say the coin flip has *one-sided error*.

2.3.2. Strongly fair coin flipping with bias p

This functionality (adapted from [7]) models a coin flip between two parties with a bounded bias. If both parties follow the protocol, they output an identical uniformly chosen bit. Even if one party does not follow the protocol, the other party outputs a random bit d that satisfies: $|\Pr[d = 0] - \Pr[d = 1]| \leq 2p$. Note that we explicitly deal with premature halting; the standard **Halt** command is *not* present in this functionality.

Functionality $\mathcal{F}^{(SCF)}$ is described as follows:

Value When this command is received for the first time from any party, $\mathcal{F}^{(SCF)}$ chooses a bit b, such that $b = 1$ with probability p and 0 with probability $1 - p$ (this bit signifies whether it will allow the adversary to set the result). If $b = 1$, the functionality sends the message **ChooseValue** to $\mathcal{A}$. Otherwise, it chooses a random bit $d \in \{0, 1\}$ and outputs (**Coin**, d) to all parties and to $\mathcal{A}$. If this command is sent more than once, all invocations but the first are ignored.

Bias d When this command is received, the functionality verifies that the sender is corrupt, that the **Value** command was previously sent by one of the parties and that $b = 1$ (if any of these conditions are not met, the command is ignored). The functionality then outputs (**Coin**, d) to all parties.

2.3.3. Bit commitment

Functionality $\mathcal{F}^{(BC)}$ (adapted from [7]) can be described as follows:

Commit b The issuer of this command is called the sender, the other party is the receiver. On receiving this command, the functionality records b and outputs **Committed** to the receiver. It then ignores any other commands until it receives the **Open** command from the sender.

Open On receiving this command from the sender, the functionality outputs (**Opened**, b) to the receiver.

2.3.4. Oblivious transfer

Functionality $\mathcal{F}^{(OT)}$ (taken from [11]) is as follows:

Send (b_0, b_1) The issuer of this command is called the sender, the other party is the receiver. On receiving this command, the functionality records (b_0, b_1) and outputs **QueryChoice** to the receiver. It then ignores all other commands until it receives a **Choice** command from the receiver.

Choice c On receiving this command from the receiver, the functionality verifies that $c \in \{0, 1\}$. It then sends b_c to the receiver.

2.4. Intermediate functionalities

In order to simplify the presentation, in the following sections we will present protocols that realize functionalities that are slightly weaker than the ones we want. We then use standard amplification techniques to construct the "full" functionalities from their weak version. In this section, we define these intermediate functionalities and state the amplification lemmas we use to construct the stronger versions of these primitives. These definitions are in the spirit of [12].

2.4.1. p-weak bit-commitment

This functionality models BC, where a corrupt receiver can cheat with probability p. Note that an ϵ-WBC protocol is a regular BC protocol when ϵ is negligible. Formally, functionality $\mathcal{F}^{(p-WBC)}$ proceeds as follows:

Commit b The issuer of this command is called the sender, the other party is the receiver. On receiving this command, the functionality records b and outputs **Committed** to the receiver. It ignores any additional **Commit** commands.

Open b On receiving this command from the sender, the functionality verifies that the sender previously sent a **Commit** b command. If so, the functionality outputs (**Opened**, b) to the receiver.

Break On receiving this command from a corrupt receiver, the functionality verifies that the sender previously sent a **Commit** b command. With probability p it sends (**Broken**, b) to the receiver and with probability $1 - p$ it sends $\perp$ to the receiver. Additional **Break** commands are ignored.

The following theorem allows us to amplify any p-WBC protocol when $p < 1$, meaning that the existence of such a protocol implies the existence of regular BC.

Theorem 2.1. *For any $p < 1$ and $\epsilon > 0$, there exists a protocol that realizes $\mathcal{F}^{(\epsilon-WBC)}$ using $O\left(\log\left(\frac{1}{\epsilon}\right)\right)$ instances of $\mathcal{F}^{(p-WBC)}$.*

The proof for this theorem is given in Section 9.3.

2.4.2. p-remotely inspectable seal

This functionality is used in our protocol for strongly fair CF. It is a strengthened version of a tamper-evident seal. With a tamper-evident seal, only its holder can interact with it. Thus, either the sender can check if it was opened, or the receiver can verify that the sealed contents were not changed, but not both at the same time. A remotely inspectable seal (RIS) is one that can be tested "remotely" (without returning it to the sender). Unfortunately, we cannot realize this "perfect" version in the DE model, therefore, relax it somewhat: we allow a corrupt receiver to learn the committed bit during the verification process and only then decide (assuming he did not previously break the seal) whether or not the verification should succeed. Our definition is actually a further relaxation[3]: the receiver may cheat with some probability: A corrupt receiver who opens the commitment before the verify stage will be caught with probability $1 - p$.

Formally, the functionality maintains an internal state variable $v = (v_b, v_s)$ consisting of the committed bit v_b and a "seal" flag v_s. It accepts the commands:

Commit b The issuer of this command is called the sender, the other party is the receiver. b can be either 0, 1. The functionality sets $v \leftarrow (b, \textbf{sealed})$. The functionality outputs **Committed** to the receiver and ignores any subsequent **Commit** commands.

Open This command is sent by the receiver. The functionality outputs (**Opened**, v_b) to the receiver. If $v_s = \textbf{sealed}$, with probability $1 - p$ the functionality sets $v_s \leftarrow \textbf{open}$.

Verify If $v_s \neq \textbf{sealed}$, the functionality outputs (**Verifying**, $\perp$) to the receiver and $\perp$ to the sender. Otherwise (no opening was detected), the functionality outputs (**Verifying**, v_b) to the receiver. If the receiver is corrupt, the functionality waits for a response. If the adversary responds with **ok**, the functionality outputs **Sealed** to the sender, otherwise it outputs $\perp$ to the sender. If the receiver is not corrupt, the functionality behaves as if the adversary had responded with **ok**. After receiving this command from the sender (and responding appropriately), the functionality ignores any subsequent **Verify** and **Open** commands.

We call 0-RIS simply "RIS". When $\epsilon$is negligible, ϵ-RIS is statistically indistinguishable from RIS. The following theorem states that a p-RIS functionality can be amplified for any $p < 1$ to get RIS:

Theorem 2.2. *For any $p < 1$ and $\epsilon > 0$, there exists a protocol that realizes $\mathcal{F}^{(RIS)}$ using $O\left(\log\left(\frac{1}{\epsilon}\right)\right)$ instances of $\mathcal{F}^{(p-RIS)}$.*

The proof for this theorem appears in Section 8.2.

[3] This second relaxation is only for convenience; we can remove it using amplification as noted in Theorem 2.2.

2.4.3. Possibly cheating weak oblivious transfer

The ideal functionality for WOT is defined in [12]. Loosely, a (p, q)-WOT protocol is a 1-2 OT protocol in which a corrupt sender gains extra information and can learn the receiver's bit with probability at most p, while a corrupt receiver gains information that allows it to learn the sender's bit with probability at most q. Here, we define a very similar functionality, (p, q)-*Possibly Cheating Weak Oblivious Transfer*.

This functionality differs from WOT in two ways: First, a corrupt sender or receiver learns whether or not cheating will be successful before committing to their bits. Second, a corrupt sender that cheats successfully is not committed to her bits — the sender can choose which bit the receiver will receive as a function of the receiver's bit.

Formally, functionality $\mathcal{F}^{(p,q-PCWOT)}$ proceeds as follows:

CanCheat When this command is first received, the functionality chooses a uniformly random number $x \in [0, 1]$ and records this number. x is returned to the issuer of the command and further **CanCheat** commands are ignored. This command can only be sent by a corrupt party.

Send (b_0, b_1) The issuer of this command is called the sender, the other party is the receiver. On receiving this command, the functionality records (b_0, b_1) and outputs **QueryChoice** to the receiver. If the receiver is corrupt and $x < q$ it also outputs (**Broken**, b_0, b_1) to the receiver. It then ignores all other commands until it receives a **Choice** command from the receiver.

Choice c On receiving this command from the receiver, the functionality verifies that $c \in \{0, 1\}$. If the sender is corrupt and $x < p$, it sends (**Broken**, c) to the sender and waits for a **Resend** command. Otherwise, it sends b_c to the receiver. Any further **Choice** commands are ignored.

Resend b On receiving this command from a corrupt sender, and if $x < p$, the functionality sends b to the receiver.

In [12], Damgård et al. prove that (p, q)-WOT implies OT iff $p + q < 1$. A careful reading of their proof shows that this is also the case for (p, q)-PCWOT, giving the following result:

Theorem 2.3. *For any $p + q < 1$ and any $\epsilon > 0$, there exists a protocol that realizes $\mathcal{F}^{(\epsilon,\epsilon-PCWOT)}$ using $O\left(\log^2\left(\frac{1}{\epsilon}\right)\right)$ instances of $\mathcal{F}^{(p,q-PCWOT)}$.*

2.5. Proofs in the UC model

Formally, the UC model defines two "worlds", which should be indistinguishable to an outside observer called the "environment machine" (denoted $\mathcal{Z}$).

The "ideal world" contains two "dummy" parties, the "target" ideal functionality, $\mathcal{Z}$ and an "ideal adversary", $\mathcal{I}$. The parties in this world are "dummy" parties because they pass any input they receive directly to the target ideal functionality, and write anything received from the ideal functionality to their local output. $\mathcal{I}$ can communicate with $\mathcal{Z}$ and the ideal functionality, and can corrupt one of the parties. $\mathcal{I}$ sees the input and any communication sent to the corrupted party, and can control the output of that party. The environment machine, $\mathcal{Z}$, can set the inputs to the parties and read their local outputs, but cannot see the communication with the ideal functionality.

The "real world" contains two "real" parties, $\mathcal{Z}$ and a "real adversary", $\mathcal{A}$. In addition, it may contain the "service" ideal functionalities (in our case, the DE functionality). $\mathcal{A}$ can communicate with $\mathcal{Z}$ and the "service" ideal functionalities, and can corrupt one of the parties. The uncorrupted parties follow the protocol, while corrupted parties are completely controlled by $\mathcal{A}$. As in the ideal world, $\mathcal{Z}$ can set the inputs for the parties and see their outputs, but not internal communication (other than what is known to the adversary).

The protocol securely realizes an ideal functionality in the UC model, if there exists $\mathcal{I}$ such that for any $\mathcal{Z}$ and $\mathcal{A}$, $\mathcal{Z}$ cannot distinguish between the ideal world and the real world. Our proofs of security follow the general outline for a proof typical of the UC model: we describe the ideal adversary, $\mathcal{I}$, that "lives" in the ideal world. Internally, $\mathcal{I}$ simulates the execution of the "real" adversary, $\mathcal{A}$. We can assume w.l.o.g. that $\mathcal{A}$ is simply a proxy for $\mathcal{Z}$, sending any commands received from the environment to the appropriate party and relaying any communication from the parties back to the environment machine. $\mathcal{I}$ simulates the "real world" for $\mathcal{A}$, in such a way that $\mathcal{Z}$ cannot distinguish between the ideal world when it is talking to $\mathcal{I}$ and the real world. In our case, we will show that $\mathcal{Z}$'s view of the execution is not only indistinguishable, but actually identical in both cases.

All the ideal adversaries used in our proofs have, roughly, the same idea. They contain a "black-box" simulation of the real adversary, intercepting its communication with the tamper-evident container functionalities and replacing it with a simulated interaction with simulated tamper-evident containers. The main problem in simulating a session that is indistinguishable from the real world is that the ideal adversary does not have access to honest parties' inputs, and so cannot just simulate the honest parties. Instead, the ideal adversary makes use of the fact that in the ideal world the "tamper-evident seals" are simulated, giving it two tools that are not available in the real world:

First, the ideal adversary does not need to commit in advance to the contents of containers (it can decide what the contents are at the time they are opened), since, in the real world, the contents of a container do not affect the view until the moment it is opened.

Second, the ideal adversary knows exactly what the real adversary is doing with the simulated containers *at the time the real adversary performs the action*, since any commands sent by the real adversary to the simulated tamper-evident container

functionality are actually received by the ideal adversary. This means that the ideal adversary knows when the real adversary is cheating. The target functionalities, when they allow cheating, fail completely if successful cheating gives the corrupt party "illegal" information: in case cheating is successful, they give the adversary the entire input of the honest party. Thus, the strategy used by the ideal adversary is to attempt to cheat (by sending a command to the target ideal functionality) when it detects the real adversary cheating. If it succeeds, it can simulate the rest of the protocol identically to a real honest party (since it now has all the information it needs). If it fails to cheat, the ideal adversary uses its "inside" information to cause the real adversary to be "caught" in the simulation.

3. Capabilities of the distinguishable weak lock model

This is the weakest of the four primitives we consider. We show that unconditionally secure BC and OT are impossible in this model. However, this model is still strictly stronger than the bare model, as weak CF is possible in this model.

3.1. A weakly fair coin flipping protocol

We give a protocol that securely realizes $\mathcal{F}^{(WCF)}$ using calls to $\mathcal{F}^{(DWL)}$. Here, Alice learns the result of the coin flip first. Note that when this protocol is implemented in the DE Model, a trivial change allows it to have one-sided error (only Bob can cheat). In this case, Bob learns the result of the coin flip first.

Protocol 3.1 (*WCF*).

1. Alice prepares and sends to Bob $4n$ containers arranged in quads. Each quad contains two containers with the value 0 and two with the value 1. The order of the 0s and 1s within the quad is random.
2. If Alice halts before completing the previous stage, Bob outputs a random bit and halts. Otherwise, Bob chooses one container from every quad and sends the chosen containers to Alice.
3. Alice verifies that all the containers Bob sent are still sealed (if not, or if Bob halts before sending all the containers, she outputs $\perp$ and halts). She then unlocks all the remaining containers, outputs the xor of the bits in the containers she received from Bob and halts.
4. Bob opens all the containers in his possession. If any triplet of open containers is improper ((0, 0, 0) or (1, 1, 1)), Bob outputs a random bit and halts. If Alice quits before unlocking the containers, Bob outputs $\perp$ and halts. Otherwise, he outputs the xor of the bits in the containers that remain in his possession and halts. In the DE model, Bob can open the containers without help from Alice, so he never outputs $\perp$.

The following theorem (whose proof appears in Section 6) states the security properties for the protocol:

Theorem 3.1. *Protocol 3.1 securely realizes $\mathcal{F}^{(WCF)}$ in the UC model.*

3.2. Oblivious transfer is impossible

Any protocol in the DWL model is also a protocol in the DE model (see Section 4). We show in Section 4.1 that OT is impossible in the DE model, hence it must also be impossible in the DWL model.

3.3. Bit-commitment is impossible

To show BC is impossible in the DWL model, we define a small set of properties that every BC protocol must satisfy in order to be considered "secure". We then show that no protocol in the DWL model can satisfy these properties simultaneously.

A BC protocol is a protocol between two players, a sender and a receiver. Formally, we can describe the protocol using four PPTs, corresponding to the commitment stage and the opening stage for each party.

$P^S_{\text{Commit}}(b, 1^n)$ receives an input bit and plays the sender's part in the commit stage of the protocol. The PPT can communicate with P^R_{Commit} and with the $\mathcal{F}^{(DWL)}$ functionality. It also has an output tape whose contents are passed to P^S_{Open}.

$P^R_{\text{Commit}}(1^n)$ plays the receiver's part in the commit stage of the protocol. It can communicate with P^S_{Commit} and with the $\mathcal{F}^{(DWL)}$ functionality. It also has an output tape whose contents are passed to P^R_{Open}.

$P^S_{\text{Open}}(1^n)$ receives the output tape of P^S_{Commit}, and can communicate with P^R_{Open} and with the $\mathcal{F}^{(DWL)}$ functionality (note that $\mathcal{F}^{(DWL)}$ retains its state between the commit and open stage).

$P^R_{\text{Open}}(1^n)$ receives the output tape of P^R_{Commit}, and can communicate with P^R_{Open} and with the $\mathcal{F}^{(DWL)}$ functionality. $P^R_{\text{Open}}(1^n)$ outputs either a bit b' or $\perp$.

A BC protocol is complete, if it satisfies:

Definition 3.2 (*Completeness*). If b is the input to P^S_{Commit}, and both parties follow the protocol, the probability that the output of $P^R_{\text{Open}}(1^n)$ is not b is a neglible function in n.

We say a BC protocol is *secure*, if it satisfies the following two properties:

Definition 3.3 (*Hiding*). Let the sender's input b be chosen uniformly at random. Then, for any adversary B substituted for P^R_{Commit} in the protocol, the probability that B can guess b is at most $\frac{1}{2} + \epsilon(n)$, where ϵ is a negligible function.

Definition 3.4 (*Binding*). For any adversary $A = (A_{\text{Commit}}, A_{\text{Open}}(x))$ substituted for P^S in the protocol, if $x \in \{0, 1\}$ is chosen independently and uniformly at random after the end of the commit stage, the probability (over A and P^R's random coins and over x) that P^R_{Open} outputs x is at most $\frac{1}{2} + \epsilon(n)$, where ϵ is a negligible function.

Implementing BC that is secure against computationally unbounded players using only the $\mathcal{F}^{(DWL)}$ functionality is impossible. We show this is the case not only for UC BC (which is a very strong notion of BC), but even for a fairly weak version: there is no BC protocol that is both unconditionally hiding and unconditionally binding in the DWL model.

Intuitively, the reason that BC is impossible is that in the DWL model the sender has access to all the information the receiver has about the sender's bit. This information cannot completely specify the bit (since in that case the hiding requirement of the commitment protocol is not satisfied), hence there must be valid decommitments for both 0 and 1 (that the receiver will accept). Since the sender knows what information the receiver has, she can determine which decommitments will be accepted (contradicting the binding requirement).

More formally, the proof proceeds in three stages. First, we show that we can assume w.l.o.g. that a BC protocol in the DWL model ends the commit phase with all containers returned to their creators. Second, we show that if the receiver is honest, the sender can compute everything the receiver knows about her bit and her random string. We then combine these facts to show that either the receiver knows her bit (hence the protocol is not hiding) or the sender can decommit to two different values (hence the protocol is not binding).

Let $P = \left(P^S_{\text{Commit}}, P^S_{\text{Open}}, P^R_{\text{Commit}}, P^R_{\text{Open}}\right)$ be a BC protocol using calls to $\mathcal{F}^{(DWL)}$, where P^S denotes the sender's side of the protocol and P^R the receiver's side. Let Alice be the sender in the commitment protocol and Bob the receiver. Denote Alice's input bit by b and her random string by r_A. Denote Bob's random string r_B and Bob's view of the protocol at the end of the commit stage V_{Bob} (w.l.o.g, this is assumed to be the output of P^R_{Commit}). We can assume w.l.o.g. that both parties know which is the final message of the commit stage (since both parties must agree at some point that the commit stage is over).

Let P' the protocol in which, at the end of the commit stage, Alice unlocks all the containers she created and Bob opens all the containers in his possession, records their contents and returns them to Alice. Formally, the protocol is defined as follows:

- P'^R_{Commit} runs P^R_{Commit} using the same input and random coins, keeping track of the locations of all containers created by the sender. When P^R_{Commit} terminates, P'^R_{Commit} waits for all containers it holds to be unlocked, then opens all of them, records their contents and returns them to P'^S.
- P'^S_{Commit} runs P^S_{Commit} using the same input and random coins, keeping track of the locations of all containers it creates. When P^S_{Commit} terminates, P'^S_{Commit} unlocks all the containers created by P^S and still held by the receiver, then waits for the containers to be returned.
- P'^S_{Open} runs P^S_{Open}, but when P^S_{Open} sends an **Unlock** command to $\mathcal{F}^{(DWL)}$ for a container that was created by P'^S_{Commit}, P'^S_{Open} instead sends a special "unlock" message to P'^R_{Open}.
- P'^R_{Open} runs P^R_{Open}, converting the special "unlock" messages sent by P'^S_{Open} to simulated **Unlocked** messages from $\mathcal{F}^{(DWL)}$. It also intercepts requests to open containers that were created by P'^S_{Commit} and simulates the responses using the recorded contents. Its output is the output of P^R_{Open}.

Lemma 3.5. *If P is both hiding and binding, so is P'.*

Proof. *P' is binding.* If P' is not binding, it means there is some adversary $A' = (A'_{\text{Commit}}, A'_{\text{Open}}(x))$ such that when A' is substituted for P'^S, P'^R will output x with probability at least $\frac{1}{2} + \text{poly}(\frac{1}{n})$.

We can construct an adversary A that will have the same probability of success when substituted for P^S in protocol P: A_{Commit} runs A'_{Commit} until P^R_{Commit} terminates, recording the contents and locations of any containers A' creates. It then continues to run A'_{Commit}, discarding any **Unlock** commands A' sends after this point, and simulating the receipt of all containers created by A' and still held by P^R (if A' asks to verify a container, A simulates an **ok** response from $\mathcal{F}^{(DWL)}$, and if it asks to open a container, A simulates the correct **Opened** response using the recorded contents).

$A_{\text{Open}}(x)$ runs $A'_{\text{Open}}(x)$. When $A'_{\text{Open}}(x)$ sends a special unlock message to P^R, $A_{\text{Open}}(x)$ sends the corresponding real unlock command to $\mathcal{F}^{(DWL)}$. Given the same input and random coins, the simulated version of P^R_{Open} under P' has a view identical to the real P^R_{Open} under P, hence the output must be the same. Therefore, the probability that A is successful is identical to the probability that A' is successful. This contradicts the hypothesis that P is binding.

P' is hiding. If P' is not hiding, there is some adversary $B' = B'_{\text{Commit}}$ that, substituted for P'^R_{Commit} in protocol P' can guess b with probability $\frac{1}{2} + \text{poly}(\frac{1}{n})$. We can construct an adversary B for the protocol P as follows: B behaves identically to B' until P^S_{Commit} terminates. It then breaks all the containers that remain in its possession and continues running B', simulating the **Unlock** messages from P^S. Since the simulation of B' under B and the real B' in protocol P' see an identical view (given the

same random coins and input), B and B' will have the same output, guessing b successfully with non-negligible advantage. This contradicts the hypothesis that P is hiding. □

Denote P'' the protocol in which, at the end of P_{Commit}, Alice returns all of Bob's containers to him and Bob uses them only in P''_{Open} (or ignores them if they are never used).

Formally, P''^S_{Commit} runs P^S_{Commit} until it terminates, keeping track of the containers created by P''^R. It then returns all those containers that it still holds to P''^R. P''^R_{Commit} runs P^R_{Commit} until it terminates, and records the ids of the containers received from P''^S_{Commit}.

P''^S_{Open} runs P^S_{Open}, replacing **Send** commands to $\mathcal{F}^{(DWL)}$ for containers sent by P''^S with special "send" messages to P''^R. When P^S_{Open} attempts to open one of the containers sent by P''^S, P''^S sends a special "return" message to P''^R and waits for it to send that container.

P''^R_{Open} runs P^R_{Open}, intercepting the special "send" and "return" messages from P''^S. In response to a "send" message it simulates a **Receipt** message from $\mathcal{F}^{(DWL)}$, and in response to a "return" message it gives the corresponding **Send** command to $\mathcal{F}^{(DWL)}$.

Lemma 3.6. *If P is both hiding and binding then so is P''.*

Proof. *P'' is binding.* Suppose P'' is not. Then, there exists some adversary $A'' = (A''_{\text{Commit}}, A''_{\text{Open}}(x))$ such that when A'' is substituted for P''^S, P''^R will output x with probability at least $\frac{1}{2} + \text{poly}(\frac{1}{n})$.

We can construct an adversary A that will have the same probability of success when substituted for P^S in protocol P: A_{Commit} runs A''_{Commit} until P^R_{Commit} terminates. It then continues to run A''_{Commit}, intercepting any **Send** commands A'' sends after this point.

$A_{\text{Open}}(x)$ runs $A''_{\text{Open}}(x)$. When $A''_{\text{Open}}(x)$ sends a special "send" message to P''^R, $A_{\text{Open}}(x)$ instead sends the corresponding real container to P^R. When A'' sends a special "return" message to P''^R, A simulates the receipt of the container from P''^R (this is possible because the container was never actually sent).

Given the same input and random coins, the simulated version of P^R_{Open} under P'' has a view identical to the real P^R_{Open} under P, hence the output must be the same. Therefore, the probability that A is successful is identical to the probability that A'' is successful. This contradicts the hypothesis that P is binding.

P'' is hiding. Suppose it is not, then there is some adversary $B'' = B''_{\text{Commit}}$ that, substituted for P'^R_{Commit} in protocol P'' can guess b with probability $\frac{1}{2} + \text{poly}(\frac{1}{n})$. We can construct an adversary B for the protocol P as follows: B runs B'' until P^S_{Commit} terminates, recording the contents and locations of containers it creates. B then simulates the receipt of all containers it created that were still held by P^S and continues running B''. If B'' tests whether a container is sealed, B simulates an **ok** response for all containers (note that since P^S is an honest party, it cannot break any lock, so this response is always correct). If B'' opens a container, B simulates the proper response using the last recorded contents for that container (since only the creator of the container can alter the contents, this response is always correct).

Given the same input and random coins, the views of B'' when P'' is running and the simulated B'' when P is running are identical, hence the output must be the same. Therefore, B can also guess b with probability $\frac{1}{2} + \text{poly}(\frac{1}{n})$, contradicting the hypothesis that P is hiding. □

Lemma 3.7. *If neither Alice (the sender) nor Bob (the receiver) break open containers (open a container that is not unlocked), Alice can compute $b, r_A \mid V_{Bob}$ (the information Bob has about b and Alice's random string at the end of the commitment phase).*

Proof. Bob's view, V_{Bob}, is composed of some sequence of the following:

1. **Seal** messages for his own containers.
2. **Receipt** messages for containers received from Alice.
3. **Send** messages for containers sent to Alice.
4. **Open** messages sent for containers he created and Alice holds (there is no point in Bob opening a container he created and also holds — he already knows what it contains).
5. **Opened** messages generated by Alice opening a container she created and he holds.
6. **Verify** messages he sent.
7. **Verified** messages received as a response to his **Verify** messages.
8. **Unlock** messages he sent.
9. Plaintext communication.

Any information Bob has about b, r_A must derive from his view of the protocol. Any messages sent by Bob do not add information about b or r_A: the contents of the message are determined solely by r_B, which is independent of b and r_A, and by the prefix of the protocol. Therefore, the **Seal**, **Send**, **Open**, **Verify** and **Unlock** messages do not contribute information about b or r_A.

The response to a **Verify** message will always be **ok**, since Alice never breaks open a container. Therefore, **Verified** messages do not contain any information about b or r_A.

It follows that all the information Bob has about b, r_A must reside in the **Receipt** and **Opened** messages and plaintext communication. However, this information is also available to Alice: Every **Receipt** message is generated by a **Send** message from Alice (so she knows the contents of all **Receipt** messages received by Bob). On the other hand, since Bob never breaks open a container, every **Open** message he sends must be preceded by an **Unlock** message from Alice. Thus, Alice must know which containers he opened (and since she created them, she knows their contents) And, of course, Alice also knows anything she sent in plaintext to Bob. □

Theorem 3.8. *$\mathcal{F}^{(BC)}$ cannot be securely realized against computationally unbounded adversaries using $\mathcal{F}^{(DWL)}$ as a primitive.*

Proof. From Lemmas 3.5 and 3.6, we can assume w.l.o.g that at the end of the Commit phase, all of Alice's containers are held by Alice and all of Bob's containers are held by Bob.

From Lemma 3.7, Alice knows everything Bob knows about b and r_A. Therefore, she can compute all the possible pairs b', r'_A which are consistent with Bob's view of the protocol.

Assume, in contradiction, that with non-negligible probability (over b and both parties' random coins), in at least poly($\frac{1}{n}$) of the pairs $b' = 0$ and in at least poly($\frac{1}{n}$) of the pairs $b' = 1$. Consider the following adversary $A = (A_{\text{Commit}}, A_{\text{Open}})$: A_{Commit} runs P^S_{Commit} with a random input b. $A_{\text{Open}}(x)$ actions depend on b:

Case 1: If $b = x$, it runs P^S_{Open}.

Case 2: if $b = 1 - x$, but in at least poly($\frac{1}{n}$) of the pairs $b' = x$, it chooses r'_A randomly from this set of pairs and simulates $P^S_{\text{Commit}}(x)$, using r'_A for the random coins, intercepting all commands to $\mathcal{F}^{(DWL)}$ but **Seal** commands and simulating the correct responses using the recorded view (note that the contents and ids of Bob's containers must be identical no matter which r'_A is chosen, because Bob's view is identical for all these pairs). A can send **Seal** commands for the containers because it currently holds all the containers it created. $A_{\text{Open}}(x)$ then runs P^S_{Open} using the output from the simulation of $P^S_{\text{Commit}}(x)$.

Case 3: If $b = 1 - x$, but only a negligible fraction of the pairs $b' = x$, it fails.

By the completeness property, the probability that P^R_{Open} outputs something other than x must be negligible in cases 1 and 2. Case 1 occurs with probability $\frac{1}{2}$ and, by our hypothesis, case 2 occurs with non-negligible probability. This contradicts the binding property of the protocol.

Assume that the probability that both $b' = 0$ and $b' = 1$ in a non-negligible fraction of the pairs is negligible. Consider the following adversary B: B_{Commit} runs P^R_{Commit}. It then outputs the majority value of b' on all the pairs b', r_A consistent with its view. By our hypothesis, with overwhelming probability $b' = b$, contradicting the hiding property of the protocol. Thus, the protocol is either not binding or not hiding. □

4. Capabilities of the distinguishable envelope model

This model is clearly at least as strong as the DWL model (defined in Section 2.2.1), since we only added capabilities to the honest players, while the adversary remains the same. In fact, we show that it is strictly stronger, by giving a protocol for BC in this model (in Section 3.3, we prove that BC is impossible in the DWL model). We also give a protocol for $\frac{1}{r}$-Strong Coin Flipping in this model and show that OT is impossible.

4.1. Oblivious transfer is impossible

Let Alice be the sender and Bob the receiver. Consider Alice's bits a_0 and a_1, as well as Bob's input c, to be random variables taken from some arbitrary distribution. Alice's view of a protocol execution can also be considered a random variable $V_A = (a_0, a_1, r_A, N_1, \ldots, N_n)$, consisting of Alice's bits, random coins (r_A) and the sequence of messages that comprise the transcript as seen by Alice. In the same way we denote Bob's view with $V_B = (c, r_B, M_1, \ldots, M_n)$, consisting of Bob's input and random coins and the sequence of messages seen by Bob.

The essence of OT (whether UC or not) is that Bob gains information about one of Alice's bits, but Alice does not know which one. We can describe the information Bob has about Alice's bits using Shannon entropy, a basic tool of information theory. The Shannon entropy of a random variable X, denoted $H(X)$ is a measure of the "uncertainty" that resides in X. When X has finite support: $H(X) = -\sum_x \Pr[X = x] \log \Pr[X = x]$.

Suppose Bob's view of a specific protocol transcript is v_B. What Bob learns about a_i ($i \in \{0, 1\}$) can be described by the conditional entropy of a_i given Bob's view of the protocol. We write this $H(a_i \mid V_B = v_B)$. If Bob knows a_i at the end of the protocol then $H(a_i \mid V_B = v_B) = 0$, since there is no uncertainty left about the value of a_i given Bob's view. If Bob has no information at all about a_i then $H(a_i \mid V_B = v_B) = 1$, since there are two equally likely values of a_i given Bob's view.

We show that in any protocol in the DE Model, Alice can calculate the amount of information Bob has about each of her bits:

Theorem 4.1. *For any protocol transcript where $V_A = v_A$ and $V_B = v_B$, both $H(a_0 \mid V_B = v_B)$ and $H(a_1 \mid V_B = v_B)$ are completely determined by v_A*

Proof. We will show how to compute $H(a_0 \mid V_B = v_B)$ using the value of V_A. Computing $H(a_1 \mid V_B = v_B)$ works in the same way, replacing a_0 with a_1.

For any injection f and any random variable, the event $Y = y$ is identical to the event $f(Y) = f(y)$. Therefore, for any two random variables X and Y, it holds that $H(X \mid Y = y) = H(X \mid f(Y) = f(y))$. We will describe an injection from V_B to a variable that Alice can (almost) compute:

1. Denote by C the set of all pairs $(id, value_{id})$ that appear in some **Opened** message from $M_1, \ldots, M_n$ and such that id is one of Alice's envelopes.
2. Denote by O the multiset of all pairs $(id, state)$ that appear in some **Verified** message from $M_1, \ldots, M_n$. This is a multiset because the same envelope may be verified multiple times. We only count the first **Verified** message after a **Receipt** message for the same envelope, however (i.e. if Bob verified the same envelope more than once without sending it to Alice between verifications, we ignore all but the first).
3. Denote M' the subsequence of the messages $M_1, \ldots, M_n$ consisting only of **Receipt** messages from $\mathcal{F}^{(DE)}$ and plaintext messages from Alice. We consider M' to contain the indices of the messages in the original sequence.

Let $f(V_B) = (O, C, c, r_B, M')$. To show that f is one-to-one, we show that given (O, C, c, r_B, M') it is possible to compute V_B by simulating Bob. The simulation proceeds as follows:

1. Run Bob (using c for the input and r_B for the random coins) until Bob either sends a message to $\mathcal{F}^{(DE)}$ or should receive a message from Alice (we an assume w.l.o.g. that Bob always knows when he is supposed to receive a message). If Bob asks to send a message to Alice the simulation pretends to have done so.
2. If Bob sends a message to $\mathcal{F}^{(DE)}$, we simulate a response from $\mathcal{F}^{(DE)}$:
   (a) If Bob sends an **Open** message for one of Alice's envelopes, we can look up the contents in C and respond with a simulated **Opened** message.
   (b) If Bob sends an **Verify** message for one of his envelopes, we can look up the result in O and respond with a simulated **Verified** message (if the envelope was verified multiple times, we return the result corresponding to the current request from the multiset, or the previous returned result if Bob did not send the envelope to Alice between verifications).
   (c) If Bob sends an **Seal** message, we store the value (and do nothing, since no response is expected).
   (d) If Bob sends an **Open** message for one of his own envelopes, we respond with an **Opened** message using the value stored earlier.
   (e) The simulation also keeps track of the locations of simulated envelopes (so that it can respond correctly if Bob tries an illegal operation, such as opening an envelope that is not in his possession).
3. If Bob should receive a message, we simulate either a plaintext message from Alice or a **Receipt** message from $\mathcal{F}^{(DE)}$ by looking it up in M'.

Given r_B, Bob is deterministic, so the simulation transcript must be identical to the original protocol view.

Finally, note that the random variables a_0 and (c, r_B) must be independent (otherwise even before beginning the protocol, Bob has information about Alice's input bits). Hence, for any random variable X: $H(a_0 \mid X, c, r_B) = H(a_0 \mid X)$. In particular, $H(a_0 \mid O, C, c, r_B, M') = H(a_0 \mid O, C, M')$.

However, Alice can compute O, C, M' from V_A: Alice can compute M' since any **Receipt** messages Bob received must have been a response to a **Send** message sent by Alice, and all messages sent by Alice (including plaintext messages) can be computed from her view of the protocol.

We can assume w.l.o.g. that Bob opens all the envelopes that remain in his possession at the end of the protocol (if the protocol is secure, the protocol in which Bob opens the envelopes at the end must be secure as well, since a corrupt Bob can always do so without getting caught). Likewise, we can assume w.l.o.g. that both players verify all of their envelopes as they are returned by the other player (again, this can be done by a corrupt player without leaking any information to the other player, so the protocol that includes this step cannot be less secure than the same protocol without it).

C consists of the contents of all of Alice's envelopes that Bob opened. Obviously, Alice knows the contents of all her envelopes (since she created them). To compute C, she only needs to know which of them were opened by Bob. Each of her envelopes is either in her possession or in Bob's possession at the end of the protocol; Alice can tell which is the case by checking if the envelope was sent more times than it was received. If it is not in her possession, she can assume Bob opened it. If it is in her possession, she verified the seal on the envelope every time it was received from Bob and the results of the verification are in her view of the protocol. If Bob opened it, at least one of the verifications must have failed. Thus, Alice can compute C. Similarly, her view tells her which of Bob's envelopes she opened and how many times each envelope was sent to Bob. Since she can assume Bob verified each envelope every time it was returned to him, she can compute the results of the **Verified** messages Bob received (and so she can compute O).

Thus, Alice can compute $H(a_0 \mid O, C, M') = H(a_0 \mid f(V_B) = f(v_B)) = H(a_0 \mid V_B = v_B)$. □

4.2. Bit commitment

In this section, we give a protocol for BC using DEs. The protocol realizes a weak version of BC (defined in Section 2.4.1). Theorem 2.1 implies that WBC is sufficient to realize "standard" BC.

Protocol 4.1 ($\frac{3}{4}$-*WBC*).

To implement **Commit** b:

1. The receiver prepares four sealed envelopes, two containing a 0 and two a 1 in random order. The receiver sends the envelopes to the sender.
2. The sender opens three envelopes (chosen randomly) and verifies that they are not all the same. Let r be the value in the remaining (sealed) envelope. The sender sends $d = b \oplus r$ to the receiver.

To implement **Open**:

1. The sender sends b and the sealed envelope to the receiver.
2. The receiver verifies that the envelope is sealed, then opens it to extract r. He verifies that $d = b \oplus r$.

The proof for the security of this protocol, stated as the following theorem, appears in Section 9:

Theorem 4.2. *Protocol 4.1 securely realizes* $\mathcal{F}^{(\frac{3}{4}-WBC)}$ *in the UC model.*

4.3. A strongly fair coin flipping protocol with bias $O(\frac{1}{r})$

The construction uses RISs (defined in Section 2.4.2), which we then show how to implement in the DE model. The idea is similar to the "standard" CF protocol using BC: Alice commits to a random bit a. Bob sends Alice a random bit b, after which Alice opens her commitment. The result is $a \oplus b$.

The reason that this is not a strongly fair CF protocol is that Alice learns the result of the toss before Bob and can decide to quit before opening her commitment. Using RIS instead of BC solves this problem, because Bob can open the commitment without Alice's help.

Ideally, we would like to replace BC with RIS (and have Alice verify that Bob did not break the seal before sending b). This almost works; If Bob quits before verification, or if the verification fails, Alice can use a as her bit, because Bob had to have decided to quit before seeing a. If Bob quits after verification (and the verification passed), Alice can use $a \oplus b$, since Bob sent b before learning a.

The reason this idea fails is that RIS allows Bob to see the committed bit *during* verification. If he does not like it, he can cause the verification to fail.

We can overcome the problem with probability $1 - \frac{1}{r}$ by doing the verification in r rounds. The trick is that Alice secretly decides on a "threshold round": after this round a failure in verification would not matter. Bob does not know which is the threshold round (he can guess with probability at most $1/r$). If Bob decides to stop before the threshold round, either he did not attempt to illegally open a commitment (in which case his decision to stop cannot depend on the result of the coin flip), or he illegally opened all the remaining commitments (opening less than that gives no information about the result). In this case, all subsequent verifications will fail, so he may as well have simply stopped at this round (note that the decision to open is made before knowing the result of the coin flip). Clearly, anything Bob does after the threshold round has no effect on the result. Only if he chooses to illegally open commitments during the threshold round can this have an effect on the outcome (since in this case, whether or not the verification fails determines whether Alice outputs a or $a \oplus b$).

The full protocol follows:

Protocol 4.2 ($\frac{1}{r}$-*SCF*). The protocol uses r instances of $\mathcal{F}^{(RIS)}$:

1. Alice chooses r random bits $a_1, \ldots, a_r$ and sends **Commit** a_i to $\mathcal{F}_i^{(RIS)}$ (this is done in parallel). Denote $a = a_1 \oplus \cdots \oplus a_r$.
2. Bob chooses a random bit b. If Alice halts before finishing the commit stage, Bob outputs b. Otherwise, he sends b to Alice.
3. If Bob halts before sending b, Alice outputs a. Otherwise, Alice chooses a secret index $j \in \{1, \ldots, r\}$.
4. The protocol now proceeds in r rounds. Round i has the following form:
   (a) Alice verifies that Bob did not open the commitment for a_i.
   (b) Bob opens the commitment for a_i (this actually occurs during the RIS verification step).
5. If the verification for round j and all preceding rounds was successful, Alice outputs $a \oplus b$. Otherwise, Alice outputs a.
6. Bob always outputs $a \oplus b$ (if Alice halts before completing the verification rounds, Bob opens the commitments himself (instead of waiting for verification).

The proof of the following theorem appears in Section 7:

Theorem 4.3. *Protocol 4.2 securely realizes* $\mathcal{F}^{(\frac{1}{r}-SCF)}$ *in the UC model.*

4.3.1. Implementation of remotely inspectable seals

We give protocol that realizes $\frac{1}{2}$-RIS. We can then apply Theorem 2.2 to amplify it to ϵ-RIS for some negligible ϵ. In addition to the $\mathcal{F}^{(DE)}$ functionality, the protocol utilizes a weak coin flip functionality with one-sided error (only Bob can cheat). This can be implemented using DEs. The WCF protocol in the DWL model, described in Section 3.1, has one-sided error in the DE model (although we do not give a formal proof in this article). Alternatively, Blum's protocol for CF also has this property, and can be implemented using BC.

Protocol 4.3 ($\frac{1}{2}$*-RIS*).

To implement **Commit** b:

1. Alice sends two envelopes, denoted e_0 and e_1 to Bob, both containing the bit b.

To implement **Verify**:

1. Alice initiates a weakly fair coin flip with Bob (the coin flip has one-sided error, so that Alice is unable to cheat).
2. Denote the result of the coin flip r. Bob opens envelope e_{1-r} and outputs (**Verifying**, b) (where b represents the contents of the envelope. Bob returns envelope e_r to Alice.
3. Alice waits for the result of the coin flip and the envelope from Bob. If the result of the coin flip is $\perp$, or if Bob does not return an envelope, Alice outputs $\perp$. Otherwise, Alice verifies that Bob returned the correct envelope and that it is still sealed. If either of these conditions are not satisfied, she outputs $\perp$, otherwise she outputs **Sealed**.

To implement **Open**:

1. Bob randomly chooses one of the envelopes in his possession. He opens the envelope and outputs (**Opened**, b) (where b is the contents of the envelope). Bob opens the other envelope as well.

The proof of the following theorem appears in Section 8.1:

Theorem 4.4. *Protocol 4.3 securely realizes* $\mathcal{F}^{(\frac{1}{2}-RIS)}$ *in the UC model.*

4.4. Lower bound for strongly fair coin flipping

In [8], Cleve proves that for any CF protocol in the standard model, one of the parties can bias the result by $\Omega(1/r)$ where r is the number of rounds. This is true even if all we allow the adversary to do is to stop early. An inspection of his proof shows that this is also true in the DE model:

Theorem 4.5. *Any r-round strongly fair CF protocol in the DE model can be biased by* $\Omega(\frac{1}{r})$.

The main idea in Cleve's proof is to construct a number of adversaries for each round of the protocol. He then proves that the average bias for all the adversaries is at least $\Omega(\frac{1}{r})$, so there must be an adversary that can bias the result by that amount. Each adversary runs the protocol correctly until it reaches "its" round. It then computes what an honest player would output had the other party stopped immediately after that round. Depending on the result, it either stops in that round or continues for one more round and then stops.

The only difficulty in implementing such an adversary in the DE model is that to compute its result it might need to open envelopes, in which case it may not be able to continue to the next round. The solution is to notice that it *can* safely open any envelopes that would not be sent to the other party at the end of the round (since it will stop in the next round in any case). Also, it must be able to compute the result without the envelopes it is about to send (since if the other party stopped after the round ends, he would no longer have access to the envelopes). Therefore, Cleve's proof is valid in the DE model as well.

5. Capabilities of the indistinguishable weak lock model

The addition of indistinguishability makes the tamper-evident seal model startlingly strong. Even in the Weak Lock variant, unconditionally secure OT is possible (and, therefore, so are BC and CF). In this section, we construct a 1–2 OT protocol using the $\mathcal{F}^{(IWL)}$ functionality. We show a $\left(\frac{1}{2}, \frac{1}{3}\right)$-PCWOT protocol (for a definition of the functionality, see 2.4.3). We can then use Theorem 2.3 to construct a full 1–2 OT protocol.

5.1. A $(\frac{1}{2}, \frac{1}{3})$-possibly cheating weak oblivious transfer protocol

The basic idea for the protocol is that the sender can encode information in the order of containers, not just in their contents. When the containers are indistinguishable, the sender can shuffle containers (thus changing the information encoded in their order) without knowing the identities of the containers themselves; this gives us the obliviousness.

In order to get a more intuitive understanding of the protocol it is useful to first consider a protocol that works only against an "honest but curious" adversary:

1. the sender prepares two containers containing the bits (0, 1), and sends them to the receiver.
2. the receiver prepares two containers of his own, also containing (0, 1). If his bit is 0, he returns both pairs to the sender with his pair first. If his bit is 1, he returns both pairs to the sender with his pair second.
3. At this point, the sender no longer knows which of the pairs is which (as long as she does not open any containers). However, she knows that both pairs contain (0, 1). She now encodes her bits, one on each pair (by leaving the pair alone for a 0 bit or exchanging the containers within the pair for a 1 bit). She returns both pairs to the receiver.
4. the receiver verifies that both his containers are still sealed and then opens them. The bit he learns from the sender can be deduced from the order of the containers in the pair. He randomly shuffles the sender's pair and returns it to the sender.
5. the sender verifies that the containers in the remaining pair are still sealed. Since the receiver shuffled the containers within the pair, the original encoded bit is lost, so the contents of the containers give her no information about the receiver's bit.

Unfortunately, this simple protocol fails when the adversary is not limited to be passive. For example, an active adversary that corrupts the receiver can replace the sender's pair of the containers with his own at stage (2). In stage (3), the sender encodes both her bits on the receiver's containers, while he still has the sender's pair to return at stage (4).

To prevent this attack, we can let the sender start with additional container pairs (say, three). Then, in stage (3), the sender can randomly choose two of her pairs and have the receiver tell her which ones they are. She can then verify that the pairs are sealed and that they are the correct ones. Now she is left with two pairs (one hers and one the receiver's), but the order may not be what the receiver wanted. So in the modified protocol, before the sender encodes her bits, the receiver tells her whether or not to switch the pairs.

If the receiver tampered with any of her pairs (or replaced them), with probability $\frac{2}{3}$ the sender will catch him (since he cannot know in advance which pairs the sender will choose to open). However, this modification gives the sender a new way to cheat: She can secretly open one of the pairs at random (before choosing which or her pairs to verify). There are four pairs, and only one is the receiver's, so with probability $\frac{3}{4}$ she chooses one of her pairs. She can then ask the receiver to give her the locations of the other two pairs. Once she knows the location of the receiver's pair, she knows which bit he wants to learn.

To counter this attack, we let the receiver add two additional pairs as well (so that he returns six pairs at stage (2)). After the sender chooses which of her pairs to verify, the receiver randomly chooses two of his pairs to verify. He gives the sender the locations and she returns the pairs to him. Since there are now six containers, three of which are the receiver's, if the sender decides to open a container she will open one of the receiver's with probability $\frac{1}{2}$ (which is allowed in a $\left(\frac{1}{2}, \frac{1}{3}\right)$-PCWOT protocol).

However, although the receiver will eventually learn that the sender cheated, if he did not catch her here (he does not with probability $\frac{1}{3}$), the sender will learn his bit before he can abort the protocol. We prevent this by having the sender choose a random value r, and encoding $a_0 \oplus r$ and $a_1 \oplus r$ instead a_0 and a_1. At the end of the protocol the receiver asks the sender to send him either r or $a_0 \oplus a_1 \oplus r$, depending on the value of his bit. Learning only one of the values encoded by the sender gives the receiver no information about the sender's bits. Given the additional information from the sender, it allows him to learn the bit he requires, but gain no information about the other bit. As long as the sender does not know which of the two encoded values the receiver learns, his request at the end of the protocol does not give her any information about his bit.

Similarly, the receiver can gain information about both of the sender's bits by opening her containers as well as his after she encodes them. This can be prevented by having the sender use the same value for both of her containers (i.e., put 1 in both containers). Since the receiver should never open the sender's pair if he follows the protocol, this should not matter. If he has not opened the pair previously, however, he now has no information about the bit encoded in the pair (since he does not know which container was originally the first in the pair).

There remains a final problem with the protocol: the receiver can cheat by lying to the sender about the locations of his pairs when he asks her to return them, and instead asking for the sender's remaining pair (along with one of his). In this case, the sender remains with two of the receiver's pairs, giving the receiver both of her bits. We solve this by having the sender randomly shuffle the pairs she returns to the receiver. If the pairs are indeed the receiver's, he can tell how she shuffled them. For the sender's pair, however, he has to guess (since he does not know their original order). This is almost enough, except that the receiver can still cheat successfully with probability $\frac{1}{2}$ by simply guessing the correct answer. To decrease the probability of successfully cheating to $\frac{1}{3}$, we use triplets instead of pairs, and require the receiver to guess the location of the second container in the triplet under the sender's permutation.

The resulting protocol is what we require. As the protocol is fairly complex, we specify separately the sender's side (Protocol 5.1a) and the receiver's side (Protocol 5.1b).

Protocol 5.1a $\frac{1}{2}, \frac{1}{3}$-PCWOT (Sender)

Input: bits a_0, a_1.

Prepare three triplets of containers. All the containers contain the value 1.
Send all nine containers to the receiver.
Wait to receive 18 containers (six triplets) from the receiver.
Select a random index $i \in_R \{1, 2, 3\}$ and send i to the receiver.
Wait to receive indices (j_1, j_2) and (k_1, k_2) from the receiver {these should be the locations of the sender's triplets (except for triplet i) and the locations of two of the receiver's triplets}.
Open all the containers in triplets j_1 and j_2 and verify that they are the correct containers.
Choose two random permutations $\pi_1, \pi_2 \in_R S_3$.
Shuffle the triplets k_1 and k_2 using π_1 and π_2, respectively.
Send the shuffled triplets k_1 and k_2 to the receiver. {the remaining unopened triplets should be the original triplet i and one of the receiver's triplets}
Wait to receive indices ℓ_1, ℓ_2 from the receiver.
Verify that $\ell_1 = \pi_1(2)$ and $\ell_2 = \pi_2(2)$. If not, abort.
Choose a random bit $r \in_R \{0, 1\}$.
if $a_0 \oplus r = 1$ **then** {Encode $a_0 \oplus r$ on first remaining triplet}
    Exchange first two containers in the first triplet. {encode a one}
else
    Do nothing. {encode a zero}
end if
if $a_1 \oplus r = 1$ **then** {Encode $a_1 \oplus r$ on second remaining triplet}
    Exchange first two containers in the second triplet. {encode a one}
else
    Do nothing. {encode a zero}
end if
Returns all six remaining containers to the receiver.
Wait to receive a bit b' from the receiver.
if $b' = 0$ **then**
    Set $x' \leftarrow r$.
else
    Set $x' \leftarrow a_0 \oplus a_1 \oplus r$.
end if
Send x' to the receiver

We prove the following theorem in Section 10:

Theorem 5.1. *Protocol 5.1 securely realizes* $\mathcal{F}^{(\frac{1}{2}, \frac{1}{3}-PCWOT)}$ *in the UC model.*

6. Proof of security for weakly fair coin flipping protocol (Protocol 3.1)

In this section, we prove Theorem 3.1. The proof follows the standard scheme for proofs in the UC model (elaborated in Section 2.5). We deal separately with the case where $\mathcal{A}$ corrupts Alice and where $\mathcal{A}$ corrupts Bob.

6.1. $\mathcal{A}$ corrupts Bob

We first describe the ideal simulator, then prove that the environment's view in the ideal and real worlds is identically distributed. The ideal simulator, $\mathcal{I}$, proceeds as follows:

1. $\mathcal{I}$ waits until ideal Alice sends a **Value** message to $\mathcal{F}^{(WCF)}$ and it receives the (**Approve**, d) message from $\mathcal{F}^{(WCF)}$. $\mathcal{I}$ now continues running the protocol with $\mathcal{A}$, simulating $\mathcal{F}^{(DWL)}$. $\mathcal{I}$ sends $4n$ **Receipt** messages to $\mathcal{A}$.
2. $\mathcal{I}$ chooses n random quads exactly as Alice would when following the protocol. Consider a quad "committed" when the contents of all unopened containers in the quad are identical (i.e., if three containers have already been opened or if two containers have been opened and contained the same value).

Protocol 5.1b $\frac{1}{2}, \frac{1}{3}$-PCWOT (Receiver)

Input: Choice bit b.

Wait to receive nine containers from the sender.
Prepare three triplets of containers (we will call them triplets 4, 5 and 6). Each triplet contains the values (0, 1, 0) in that order.
Choose a random permutation $\sigma \in S_6$.
Shuffle all six triplets using σ. {the three containers in each triplet are not shuffled}
Send all 18 containers to the sender.
Wait to receive an index i from the sender.
Send the indices $\sigma(\{1, 2, 3\} \setminus \{i\})$ and $\sigma(\{5, 6\})$to the sender. {the locations of the sender's triplets except for triplet i and the locations of the last two triplets created by the receiver}.
Wait to receive two triplets from the sender.
Verifies that all the containers in the received triplets were unopened and that they are from the original triplets 5 and 6.
Open the containers. Let ℓ_1, ℓ_2 be the index of the container containing 1 in each triplet. Send ℓ_1, ℓ_2 to the sender. {e.g., ℓ_1 should be $\pi_1(2)$}
Wait to receive six containers (two triplets) from the sender.
if $\sigma(i) > \sigma(4)$ **then**
    Verify that all the containers in the first triplet are sealed and were originally from triplet 4. If not, abort.
    Let $x = 1$ iff the first container in the first triplet contains 1. $\{x = a_0 \oplus r = 1\}$
    Set $c \leftarrow 0$
else
    Verify that all the containers in the second triplet are sealed and were originally from triplet 4. If not, abort.
    Let $x = 1$ iff the first container in the second triplet contains 1. $\{x = a_1 \oplus r = 1\}$
    Set $c \leftarrow 1$
end if
Send $b \oplus c$ to the sender.
Wait to receive response x' from the sender.
Output $x \oplus x'$. $\{x \oplus x' = a_b\}$

3. As long as there is at least one uncommitted quad, $\mathcal{I}$ responds to **Open** messages from $\mathcal{A}$ by returning the values chosen in stage (2).
4. When only a single uncommitted quad remains, denote by x the xor of the values for the committed quads. $\mathcal{I}$ will force the last unopened container in the quad to have the value $x \oplus d$, by choosing the responses from the distribution of permutations conditioned on the last container having the forced value.
5. $\mathcal{I}$ waits for $\mathcal{A}$ to return one container from each quad.
6. If $\mathcal{A}$ halts before returning n containers, or if any of the n containers was opened, $\mathcal{I}$ sends a **Halt** command to $\mathcal{F}^{(WCF)}$. Otherwise, it sends a **Continue** command.
7. $\mathcal{I}$ simulates the **Unlocked** messages for all the containers still held by $\mathcal{A}$. It continues the simulation until $\mathcal{A}$ halts.

Lemma 6.1. *For any $\mathcal{Z}$ and $\mathcal{A}$, when $\mathcal{A}$ corrupts Bob, $\mathcal{Z}$'s view of the simulated protocol in the ideal world and $\mathcal{Z}$'s view in the real world are identically distributed.*

Proof. $\mathcal{Z}$'s view of the protocol in both worlds is identical, except for the contents of the containers sent by Alice. An inspection of the simulation shows that the distribution of the contents is also identical: in both the real and ideal worlds, the contents of each quad are uniformly random permutations of (0, 0, 1, 1). Also in both cases, the xor of the committed value of all the quads is a uniformly random bit b. If $\mathcal{A}$ does not open more than three containers in any quad, and returns containers according to the protocol, this is the bit output by Alice in both the real and ideal worlds. If $\mathcal{A}$ opens all four containers, or does not return them according to the protocol, Alice will output $\perp$ in both the real and ideal worlds. □

6.2. $\mathcal{A}$ corrupts Alice

As in the previous case, we first describe the ideal simulator, then prove that the environment's view in the ideal and real worlds is identically distributed. The ideal simulator, $\mathcal{I}$, proceeds as follows:

1. $\mathcal{I}$ sends a **Value** message to $\mathcal{F}^{(WCF)}$ and waits to receive the (**Approve**, d) message from $\mathcal{F}^{(WCF)}$.
2. $\mathcal{I}$ waits for $\mathcal{A}$ to send the $4n$ **Seal** and **Send** messages to $\mathcal{F}^{(WCF)}$.

Case 2.1: If at least one of the quads is proper (i.e., contains two 0s and two 1s), $\mathcal{I}$ chooses which containers to send in the other quads randomly, and then chooses a container to send in the proper quad so that the xor of all the sent containers is d.

Case 2.2: If all the quads are improper, $\mathcal{I}$ chooses the containers to send from the uniform distribution conditioned on the event that at least one quad has three remaining containers that contain identical bits.

3. $\mathcal{I}$ sends the chosen containers to $\mathcal{A}$, and waits for $\mathcal{A}$ to unlock the remaining containers.
4. If $\mathcal{A}$ does not unlock all the containers, or if one of the remaining quads is improper, $\mathcal{I}$ sends a **Halt** command to $\mathcal{F}^{(WCF)}$. Otherwise $\mathcal{I}$ sends a **Continue** command to $\mathcal{F}^{(WCF)}$.

Lemma 6.2. *For any $\epsilon > 0$ there exists $n = O(\log \frac{1}{\epsilon})$, such that for any $\mathcal{Z}$ and $\mathcal{A}$, when $\mathcal{A}$ corrupts Alice the statistical distance between $\mathcal{Z}$'s view of the simulated protocol in the ideal world and $\mathcal{Z}$'s view in the real world is less than ϵ.*

Proof. $\mathcal{Z}$'s view of the protocol in both worlds is identical, except for the choice of containers sent by Bob. In the real world, Bob's choices are always uniformly random. If not all quads are improper, the distribution of Bob's choices in the ideal world is also uniformly random (since d is uniformly random, and the only choice made by $\mathcal{I}$ that is not completely random is to condition on the xor of the quad values being d). If all the quads are improper, the statistical difference between the uniform distribution and $\mathcal{I}$'s choices is exponentially small in n, since each quad has three remaining identical containers with probability at least $\frac{3}{4}$, and the events for each quad are independent (thus the probability that none of the quads was bad is at most $(\frac{3}{4})^n$). □

7. Proof of security for strongly fair coin flip protocol (Protocol 4.2)

In this section, we prove Theorem 4.3. The proof follows the standard scheme for proofs in the UC model (elaborated in Section 2.5). We deal separately with the case where $\mathcal{A}$ corrupts the sender and where $\mathcal{A}$ corrupts the receiver.

7.1. $\mathcal{A}$ corrupts Alice

1. $\mathcal{I}$ sends a **Value** command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$. If it receives a **ChooseValue** message from $\mathcal{F}^{(\frac{1}{r}-SCF)}$ it randomly chooses a bit d and sends a **Bias** d command. Denote by d the result of the coin flip.
2. $\mathcal{I}$ waits for $\mathcal{A}$ to commit to the bits $a_1, \ldots, a_r$. If $\mathcal{A}$ stops before committing to r bits, $\mathcal{I}$ halts as well.
3. Otherwise, $\mathcal{I}$ simulates Bob sending $b = d \oplus a_1 \oplus \cdots \oplus a_r$ to Alice. $\mathcal{I}$ then continues the protocol with the simulated Bob behaving honestly.

Lemma 7.1. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only Alice, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. The proof is by inspection. First, note that the output of the ideal Bob always matches the output of the simulated Bob (by the choice of b). Since $\mathcal{I}$ simulates Bob following the protocol precisely, the only difference $\mathcal{Z}$ could notice is the distribution of b. However, this is uniform in both the real and ideal worlds, since in the ideal world d (the result of $\mathcal{F}^{(\frac{1}{r}-SCF)}$'s coin flip) is uniformly distributed, and in the real world Bob chooses b uniformly. Thus, $\mathcal{Z}$'s view is identically distributed in both worlds. □

7.2. $\mathcal{A}$ corrupts Bob

1. $\mathcal{I}$ sends a **Value** command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$. We will say that $\mathcal{I}$ "has control" if it received a **ChooseValue** message, and that $\mathcal{I}$ "does not have control" if it received a '(**Coin**, d) message from $\mathcal{F}^{(\frac{1}{r}-SCF)}$.
2. If $\mathcal{I}$ has control, it chooses a random bit d itself.
3. $\mathcal{I}$ simulates Bob receiving commit messages from $\mathcal{F}_1^{(RIS)}, \ldots, \mathcal{F}_r^{(RIS)}$.
4. $\mathcal{I}$ waits for Bob (controlled by $\mathcal{A}$) to send b to Alice.

Case 1: If $\mathcal{A}$ halts before sending b, $\mathcal{I}$ sends a **Bias** d command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$ and also halts.

Case 2: If $\mathcal{A}$ attempts to open the commitments before sending b, or if $b = 0$, $\mathcal{I}$ sends a **Bias** d command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$ (this is ignored if $\mathcal{I}$ does not have control). $\mathcal{I}$ then randomly chooses $a_2, \ldots, a_r$, sets $a_1 \leftarrow d \bigoplus_{i>1} a_i$ and continues the protocol, proceeding as if Alice sent **Commit** a_i to $\mathcal{F}_i^{(RIS)}$. In this case, no matter what Bob does, in the real-world protocol Alice must eventually output d.

Case 3: If $\mathcal{A}$ sends $b = 1$ before opening any commitments:

i. $\mathcal{I}$ begins simulating the protocol rounds, randomly choosing a value for each a_i when $\mathcal{A}$ opens (or simulated Alice verifies) $\mathcal{F}_i^{(RIS)}$. The simulation continues in this manner until the contents of all but one of the commitments have been revealed (either because $\mathcal{A}$ prematurely opened the commitments, or during the verification phase).

ii. Call a round j "good" if the verification stage of round j succeeded and all previous rounds were good. Denote the current round by i, the index of the highest good round so far by j (by definition $j < i$), and by k the smallest index such that the committed bit in instance $\mathcal{F}_k^{(RIS)}$ is not yet known to $\mathcal{A}$ (note that $k \geq i$, since all instances up to i must have been revealed during verification). The actions of $\mathcal{I}$ now depend on i, j, k and whether $\mathcal{I}$ has control:

Case 3.1: If $i < k$, or if $\mathcal{F}_k^{(RIS)}$ is being opened (rather than verified): (this is equivalent to the case where even in the real world $\mathcal{A}$ could not bias the result)
- $\mathcal{I}$ sends a **Bias** d command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$.
- $\mathcal{I}$ chooses a random index $i^* \in \{1, \ldots, r\}$.
- If $i^* > j$, $\mathcal{I}$ sets $a_k \leftarrow d \bigoplus_{\ell \neq k} a_\ell$, otherwise $a_k \leftarrow b \oplus d \bigoplus_{\ell \neq k} a_\ell$.
- $\mathcal{I}$ continues the simulation as if Alice had actually chosen the bits $a_1, \ldots, a_r$ to commit and the secret threshold round i^*. Note that if Alice had actually followed the protocol, the choice of a_k ensures that she always outputs d. This is because round i will certainly fail verification (since $\mathcal{F}_i^{(RIS)}$ has already been opened), so round j will remain the last round which passed verification.

Case 3.2: If $i = k$, $\mathcal{F}_k^{(RIS)}$ is being verified and $\mathcal{I}$ does not have control: (this is equivalent to the case where $\mathcal{A}$ did not correctly guess the secret threshold round, but could have cheated successfully if he had)
- $\mathcal{I}$ chooses a random index $i^* \in \{1, \ldots, r\} \setminus \{i\}$.
- If $i^* > j$, $\mathcal{I}$ sets $a_k \leftarrow d \bigoplus_{\ell \neq k} a_\ell$, otherwise $a_k = b \oplus d \bigoplus_{\ell \neq k} a_\ell$.
- $\mathcal{I}$ continues the simulation as if Alice had actually chosen the bits $a_1, \ldots, a_r$ to commit and the secret threshold round i^*. Note that if Alice had actually followed the protocol, the choice of a_k ensures that she always outputs d. This is because, by the choice of i^*, it does not matter whether or not round i fails verification (either $i^* > j$, in which case also $i^* > i$, or $i^* \leq j < i$).

Case 3.3: If $i = k$, $\mathcal{F}_k^{(RIS)}$ is being verified and $\mathcal{I}$ has control: (this is equivalent to the case where $\mathcal{A}$ correctly guessed the secret threshold i, and can cheat successfully)
- $\mathcal{I}$ chooses a random bit for a_k and continues the simulation.
- If $\mathcal{A}$ chooses to fail the verification, $\mathcal{I}$ sets $d^* \leftarrow d \bigoplus_\ell a_\ell$, otherwise (the verification succeeds) $\mathcal{I}$ sets $d^* \leftarrow b \oplus d \bigoplus_\ell a_\ell$.
- $\mathcal{I}$ sends a **Bias** d^* command to $\mathcal{F}^{(\frac{1}{r}-SCF)}$.
- $\mathcal{I}$ continues the simulation until $\mathcal{A}$ halts.

Lemma 7.2. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only Bob, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. $\mathcal{Z}$'s view can consist of $a_1, \ldots, a_r$ (the results of opening the commitments) and of the ideal Alice's output d.

In both the real and ideal worlds, in all cases the first $r - 1$ commitments opened by $\mathcal{A}$ are independent and uniformly random (this can be easily seen by inspecting the simulator).

For any adversary that reaches Case 1 or Case 2 in the real world, the final commitment is always the xor of b (the bit sent by $\mathcal{A}$), the first $r - 1$ commitments and the output of the real Alice (since the threshold round does not affect the result in this case). This is also the situation in the ideal world.

For an adversary that reaches Case 3.1, the final commitment is the xor of the first $r - 1$ commitments and the output of the real Alice with probability $\frac{r-j}{r}$ (this is the probability that the secret threshold round was after the last good round), and the complement of that with probability $\frac{j}{r}$ (the probability that the threshold round is in the first j rounds). By the choice of i^*, the distribution of the last commitment in the ideal model is identical in this case.

Finally, consider the adversary that reaches Case 3.2 or Case 3.3. This adversary is honest until round i, then opens all commitments except $\mathcal{F}_i^{(RIS)}$, whose contents are revealed during verification.

1. In the real world, with probability $\frac{1}{r}$ round i is the threshold round, in which case the final commitment is the xor of the first $r - 1$ commitments and d if $\mathcal{A}$ fails the verification and the complement of that if $\mathcal{A}$ does not fail. With the same probability, $\mathcal{I}$ is in control, and therefore executes Case 3.3 (which calculates the final commitment in the same way).
2. With probability $1 - \frac{1}{r}$, round i is not the threshold round. In this case, the final commitment is the xor of the first $r - 1$ commitments and d with probability $\frac{r-i}{r-1}$ (the threshold round is after i), and the complement of that with probability $\frac{i-1}{r-1}$ (the threshold round is before i). In the same way, with probability $1 - \frac{1}{r}$, $\mathcal{I}$ is not in control, and executes Case 3.2. The choice of i^* ensures the correct distribution of the final commitment.

Since any adversary must reach one of the cases above, we have shown that for all adversaries $\mathcal{Z}$'s view of the protocol is identical in the real and ideal worlds. □

Together, Lemmas 7.1 and 7.2 imply Theorem 4.3.

8. Proof of security for remotely inspectable seals

Below, we prove Theorem 4.4 (in Section 8.1) and Theorem 2.2 (in Section 8.2).

8.1. Proof of security for $\frac{1}{2}$-RIS protocol (Protocol 4.3)

The proof of Theorem 4.4 follows the standard scheme for proofs in the UC model (elaborated in Section 2.5). We deal separately with the case where $\mathcal{A}$ corrupts the sender and where $\mathcal{A}$ corrupts the receiver.

8.1.1. $\mathcal{A}$ corrupts Alice (the sender)

To simulate the **Commit** command, $\mathcal{I}$ waits until $\mathcal{A}$ sends two envelopes to Bob. Denote the envelopes e_0 and e_1.

Case 1: If $\mathcal{A}$ does not send the envelopes, $\mathcal{I}$ sends the **Halt** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ (causing ideal Bob to output $\perp$) and halts.

Case 2: If both envelopes contained the same bit b, $\mathcal{I}$ sends a **Commit** b message to $\mathcal{F}^{(\frac{1}{2}-RIS)}$.

Case 3: If the envelopes contained two different bits, $\mathcal{I}$ randomly selects a bit b and sends **Commit** b to $\mathcal{F}^{(\frac{1}{2}-RIS)}$.

To simulate the **Verify** command:

1. $\mathcal{I}$ waits for $\mathcal{A}$ to initiate a coin flip.
2. If both envelopes sent by $\mathcal{A}$ contained the same bit, $\mathcal{I}$ chooses a random bit r, otherwise it sets r to the index of the envelope containing b.
3. $\mathcal{I}$ sends r as the result of the coin flip to $\mathcal{A}$.
4. $\mathcal{I}$ simulates sending envelope e_r to $\mathcal{A}$.
5. $\mathcal{I}$ sends the **Verify** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ and waits for the functionality's response.
   Case 1: If the response is $\perp$, verifying envelope e_r will return a **broken** message.
   Case 2: If the response was **Sealed**, verifying envelope e_r will return a **sealed** message.
6. $\mathcal{I}$ continues the simulation until $\mathcal{A}$ halts.

Note that the **Open** command need not be simulated in this case — in both the ideal and the real worlds this does not involve the sender at all.

Lemma 8.1. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only Alice, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. The proof is by case analysis. First, consider the view during the commit stage. Any adversary must fall in one of the three cases. In Case 1, in both the real and ideal worlds $\mathcal{Z}$'s view consists of Bob outputting $\perp$ and Alice halting. In Case 2 and Case 3, $\mathcal{Z}$'s view looks the same from $\mathcal{A}$'s point of view, and in both worlds Bob will output **Committed**.

If $\mathcal{Z}$ tells Bob to open the commitment before the verify stage, The output will be identical in the real and ideal worlds (it will be (**Opened**, b), where b is a uniformly random bit if $\mathcal{A}$ committed two different bits).

During the verification stage, r is always a random uniform bit. There are only two cases to consider: either $\mathcal{Z}$ told Bob to open the commitment earlier, or it did not. If it did, $\mathcal{F}^{(\frac{1}{2}-RIS)}$ will return a failed verification, and $\mathcal{A}$ will also see a failed verification (exactly as would be the case in the real world). If it did not, $\mathcal{A}$ will see a successful verification in both the real and ideal worlds.

Thus, in all cases $\mathcal{Z}$'s view is identically distributed in both worlds. □

8.1.2. $\mathcal{A}$ corrupts Bob (the receiver)

The simulation is in two phases. In the initial phase (corresponding to the **Commit** and **Open** commands):

1. $\mathcal{I}$ waits until it receives **Committed** from $\mathcal{F}^{(\frac{1}{2}-RIS)}$. It then simulates $\mathcal{A}$ receiving two envelopes, e_0 and e_1.
2. If $\mathcal{A}$ requests to open any of the envelopes, $\mathcal{I}$ sends an **Open** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ and waits to receive the (**Opened**, b) response. It then continues the simulation as if both envelopes had contained b.

The second phase begins when $\mathcal{I}$ receives a (**Verifying**, x) message from $\mathcal{F}^{(\frac{1}{2}-RIS)}$ (signifying that ideal Alice sent a **Verify** command). $\mathcal{I}$ initiates the verification phase with $\mathcal{A}$.

1. $\mathcal{I}$ chooses r in the following way: If, in the verification message, $x \neq \perp$ (that is, $\mathcal{I}$ has a choice about whether the verification will fail), it chooses r randomly from the set of unopened envelopes (if both were opened, it chooses randomly between them). If, in the verification message, $x = \perp$ (that is, the verification will definitely fail), $\mathcal{I}$ chooses r randomly from the set of opened envelopes (note that at least one envelope must be open for this to occur, because otherwise $\mathcal{I}$ would not have sent an **Open** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ and would thus always have a choice).
2. $\mathcal{I}$ continues the simulation following the protocol exactly, letting the contents of the envelopes both be b (where $b \leftarrow x$ if $x \neq \perp$), otherwise it is the response to the **Open** command sent in the previous phase.
3. The simulation continues until $\mathcal{A}$ returns an envelope. If that envelope was opened, or its index does not match r, $\mathcal{I}$ fails the verification by sending a **Halt** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$. If the envelope was not opened and its index does match r, $\mathcal{I}$ sends the **ok** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$ (note that if $\mathcal{I}$ had no choice, the index r always matches an envelope that was already opened).

Lemma 8.2. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only Bob, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. Since $\mathcal{I}$ simulates Alice exactly, except for the contents of the envelopes and the result of the coin flip and her response to verification, these are the only things that can differ in $\mathcal{Z}$'s view between the real and ideal worlds.

Simple inspection of the protocol shows that ideal Alice's output and the contents of the envelopes are always consistent with $\mathcal{A}$'s view. It remains to show that the distribution of r is identical in the real and ideal worlds. The only case in the ideal world in which r is not chosen uniformly at random by $\mathcal{I}$ is when exactly one of the envelopes was opened. However, this means $\mathcal{I}$ must have sent an **Open** command to $\mathcal{F}^{(\frac{1}{2}-RIS)}$, and therefore with probability $\frac{1}{2}$ the verification will fail. Thus, r is still distributed uniformly in this case. □

Together, Lemmas 8.2 and 8.1 prove Theorem 4.4.

8.2. Amplification for remotely inspectable seals

The following protocol constructs an p^k-RIS using k instances of $\mathcal{F}^{(p-RIS)}$:

Commit b Alice chooses k random values $r_1, \ldots, r_k$ such that $r_1 \oplus \cdots \oplus r_k = b$. She commits to the values in parallel using $\mathcal{F}_1^{(p-RIS)}, \ldots, \mathcal{F}_k^{(p-RIS)}$.

Verify Alice sends **Verify** commands in parallel to all k instances of $\mathcal{F}^{(p-RIS)}$. The verification passes only if all k verifications return **Sealed**.

Open Bob opens all k commitments. The result is the xor of the values returned.

The ideal adversary in this case is fairly simple. The case where the sender is corrupt is trivial, and we omit it (since the sender cannot cheat in the basic $\mathcal{F}^{(p-RIS)}$ instance). When $\mathcal{A}$ corrupts the receiver, the simulation works in two phases: In the initial phase (corresponding to **Commit** and **Open**):

1. $\mathcal{I}$ waits to receive the **Committed** command from $\mathcal{F}^{(p^k-RIS)}$.
2. Whenever $\mathcal{A}$ asks to open a commitment for $\mathcal{F}_i^{(p-RIS)}$:
   - Case 2.1: If at least one additional commitment is still unopened, $\mathcal{I}$ chooses a random bit r_i and returns this as the committed value.
   - Case 2.2: If $\mathcal{F}_i^{(p-RIS)}$ is the last unopened $\mathcal{F}^{(p-RIS)}$ instance, $\mathcal{I}$ sends an **Open** command to $\mathcal{F}^{(p^k-RIS)}$ and sets the value of the last commitment to be the xor of all the other commitments and the response, b.

The second phase begins when $\mathcal{I}$ receives a (**Verifying**, x) message from $\mathcal{F}^{(p^k-RIS)}$ (signifying that ideal Alice sent a **Verify** command). $\mathcal{I}$ initiates the verification phase with $\mathcal{A}$. Denote the number commitments opened by $\mathcal{A}$ by j.

Case 1: If $j = k$, $\mathcal{I}$ has sent an **Open** command previously to $\mathcal{F}^{(p^k-RIS)}$.
- Case 1.1: If it has a choice about verification (occurs with probability p^k), $\mathcal{I}$ sends a (**Verifying**, r_i) message to $\mathcal{A}$ for all instances of $\mathcal{F}^{(p-RIS)}$. If $\mathcal{A}$ decides to fail verification in any of the instances, $\mathcal{I}$ sends a **Halt** command to $\mathcal{F}^{(p^k-RIS)}$. Otherwise, $\mathcal{I}$ sends an **ok** response to $\mathcal{F}^{(p^k-RIS)}$.
- Case 1.2: Otherwise, $\mathcal{I}$ chooses k bits $q_1, \ldots, q_k$ by sampling from the binomial distribution $B(k, p)$, conditioned on at least one bit being 1 (i.e., equivalent to letting $q_i = 1$ independently with probability p, repeating until not all bits are 0). For each bit where $q_i = 0$ it sends (**Verifying**, r_i), and for the other bits it sends (**Verifying**, $\perp$). $\mathcal{I}$ sends a **Halt** command to $\mathcal{F}^{(p^k-RIS)}$.

Case 2: If $j < k$, no **Open** command was sent, so $\mathcal{I}$ will always have a choice whether to fail verification. $\mathcal{I}$ sends a (**Verifying**, x_i) message to $\mathcal{A}$ for each instance of $\mathcal{F}^{(p-RIS)}$. For instances which were not opened, $x_i = r_i$. For instances that were opened, $\mathcal{I}$ chooses with probability p to send $x_i = r_i$ and with probability $1 - p$ to send $x_i = \perp$. It then waits for $\mathcal{A}$ to respond. If in any of the instances it chooses $x_i = \perp$, or if $\mathcal{A}$ decides to fail verification in any of the instances, it sends a **Halt** command to $\mathcal{F}^{(p^k-RIS)}$. Otherwise, $\mathcal{I}$ sends an **ok** response to $\mathcal{F}^{(p^k-RIS)}$.

It is easy to see by inspection that the adversary's view is identical in the real and ideal worlds. Setting $k = O(\log \frac{1}{\epsilon})$, the amplification protocol gives us the proof for Theorem 2.2.

9. Proof of security for bit-commitment protocol

In this section, we prove Protocol 4.1 realizes the WBC functionality (proving Theorem 4.2) and show how to amplify WBC to get full BC (proving Theorem 2.1). We begin with the proof of security for Protocol 4.1. The proof follows the standard scheme for proofs in the UC model (elaborated in Section 2.5). We deal separately with the case where $\mathcal{A}$ corrupts the sender and where $\mathcal{A}$ corrupts the receiver.

9.1. $\mathcal{A}$ corrupts Alice (the sender)

We divide the simulation, like the protocol, into two phases.

*9.1.1. Simulation of the **Commit** phase*

$\mathcal{I}$ starts the simulated commit protocol with $\mathcal{A}$ ($\mathcal{I}$ simulates the honest receiver, Bob, in this protocol). $\mathcal{I}$ sends four (simulated) envelopes to $\mathcal{A}$. $\mathcal{I}$ chooses a random permutation $\sigma \in S_4$. If $\mathcal{A}$ opens any of the envelopes, $\mathcal{I}$ gives results that are consistent with Bob following the protocol (i.e., the envelopes' contents are determined by $\sigma(0, 0, 1, 1)$). $\mathcal{I}$ continues the simulation until Alice (controlled by $\mathcal{A}$) sends a bit, d, to Bob (as required by the protocol). The succeeding actions depend on how many envelopes $\mathcal{A}$ opened:

Case 1: $\mathcal{A}$ did not open any envelope or opened two envelopes containing different bits. In this case, $\mathcal{I}$ chooses a random bit b and sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

Case 2: $\mathcal{A}$ opened a single envelope containing x. In this case, $\mathcal{I}$ chooses a random bit b to be $d \oplus x$ with probability $\frac{1}{3}$ and $d \oplus (1 - x)$ with probability $\frac{2}{3}$. $\mathcal{I}$ sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

Case 3: Alice opened two envelopes containing identical bits x. Letting $b = d \oplus (1 - x)$, $\mathcal{I}$ sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

Case 4: Alice opened three envelopes whose xor is x. Letting $b = d \oplus (1 - x)$, $\mathcal{I}$ sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

Case 5: Alice opened four envelopes. Letting $b = 0$, $\mathcal{I}$ sends a **Commit** b command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$.

*9.1.2. Simulation of the **Open** phase*

$\mathcal{I}$ begins simulating the *Open* phase of the protocol with $\mathcal{A}$, and waits for $\mathcal{A}$ to send an envelope and a bit b'. If $\mathcal{A}$ asks to open an envelope i before this occurs, $\mathcal{I}$ proceeds in the following way:

Let $P_{consistent}$ be the set of permutations of (0, 0, 1, 1) that are consistent with $\mathcal{A}$'s view so far (i.e., the permutations that map the correct contents to the envelopes $\mathcal{A}$ has already opened), and P_{valid} the set of permutations in which at least one of the envelopes that will remain unopened after opening i contains $b \oplus d$ (where b is the bit to which $\mathcal{I}$ committed in the *Commit* phase). $\mathcal{I}$ randomly chooses a permutation from $P_{consistent} \cap P_{valid}$ and responds to the request to open i as if Bob had chosen this permutation in the *Commit* phase.

Note that $\mathcal{I}$'s choice of d and b ensures that at the end of the *Commit* phase $P_{consistent} \cap P_{valid}$ is not empty. As long as i is not the last unopened envelope, $P_{consistent} \cap P_{valid}$ will remain non-empty. If i is the last unopened envelope, $\mathcal{I}$ responds with the value consistent with the other opened envelopes.

Once $\mathcal{A}$ sends the bit b' and an envelope, $\mathcal{I}$ proceeds as follows: If the envelope is unopened, and $b' = b$, $\mathcal{I}$ sends the **Open** command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$. Otherwise, $\mathcal{I}$ aborts the protocol by sending the **Halt** command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$ (and simulating Bob aborting the protocol to $\mathcal{A}$).

Lemma 9.1. *For any environment machine $\mathcal{Z}$ and any real adversary $\mathcal{A}$ that corrupts only the sender, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. $\mathcal{I}$ simulates Bob (the receiver) exactly following the protocol (apart from the envelope contents), and the simulation ensures that the ideal Bob's output is consistent with $\mathcal{A}$'s view of the protocol. The only possible differences between $\mathcal{Z}$'s view in the real and ideal worlds are the contents of the envelopes sent by Bob. Inspection of the protocol and simulation shows that in both the real and ideal worlds, $\mathcal{A}$ always sees a random permutation of (0, 0, 1, 1). □

9.2. $\mathcal{A}$ corrupts Bob (the receiver)

As before, the simulation is divided into two phases.

*9.2.1. Simulation of the **Commit** phase*

$\mathcal{I}$ waits until $\mathcal{A}$ sends four envelopes and until the **Committed** message is received from $\mathcal{F}^{(\frac{3}{4}-WBC)}$. $\mathcal{I}$'s actions depend on the contents of the envelopes sent by $\mathcal{A}$:

Case 1: If the envelopes sent by $\mathcal{A}$ are a valid quad (two zeroes and two ones), $\mathcal{I}$ sends a random bit d to $\mathcal{A}$.

Case 2: If the envelopes are all identical (all zeroes or all ones), $\mathcal{I}$ aborts the protocol by sending the **Halt** command to $\mathcal{F}^{(\frac{3}{4}-WBC)}$ (and simulating Alice aborting the protocol to $\mathcal{A}$).

Case 3: If the envelopes contain three ones and a zero, or three zeroes and a one, denote x the singleton bit. $\mathcal{I}$ sends a **Break** message to $\mathcal{F}^{(\frac{3}{4}-WBC)}$. If the response is $\perp$, $\mathcal{I}$ simulates Alice aborting the protocol to $\mathcal{A}$ and halts. If the response is (**Broken**, b), $\mathcal{I}$ sends $b \oplus (1 - x)$ to $\mathcal{A}$.

*9.2.2. Simulation of the **Open** phase*

$\mathcal{I}$ waits to receive the (**Opened**, b) message from $\mathcal{F}^{(\frac{3}{4}-WBC)}$. It then proceeds depending on $\mathcal{A}$'s actions in the *Commit* phase:

Case 1: If $\mathcal{A}$ sent a valid quad, $\mathcal{I}$ randomly picks one of the two envelopes that contain $d \oplus b$ and returns it to $\mathcal{A}$.
Case 2: If the envelopes sent by $\mathcal{A}$ were not a valid quad, they must be three ones and a zero or three zeroes and a one (otherwise $\mathcal{I}$ would have aborted in the *Commit* phase). In this case, $\mathcal{I}$ randomly chooses one of the three identical envelopes and simulates returning it to $\mathcal{A}$.

$\mathcal{I}$ sends the bit b to $\mathcal{A}$ as well. If $\mathcal{A}$ checks whether the envelope returned by Alice is sealed, $\mathcal{I}$ simulates an affirmative reply from $\mathcal{F}^{(DE)}$.

Lemma 9.2. *For any environment machine $\mathcal{Z}$ and any real adversary $\mathcal{A}$ that corrupts only the receiver, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. $\mathcal{I}$'s simulation of Alice (the sender) is always consistent with a real Alice that follows the protocol (from $\mathcal{A}$'s point of view), and it ensures that the ideal Alice's output is also consistent with $\mathcal{A}$'s view. $\mathcal{A}$'s view consists of d, the bit sent by Alice in the commit phase (or Alice halting in the commit phase), and the choice of envelope returned in the open phase. In both the real and ideal worlds, when $\mathcal{A}$ sends a proper quad d is uniformly random. When $\mathcal{A}$ sends a quad whose bits are all identical, in both worlds Alice will abort. When $\mathcal{A}$ sends a quad containing three bits with value $1 - x$ and one bit with value x, in the real world Alice would abort with probability $\frac{1}{4}$ (if x is the unopened envelope), and send $d = b \oplus (1 - x)$ with probability $\frac{3}{4}$. In the ideal world, d is distributed identically, since $\mathcal{F}^{(\frac{3}{4}-WBC)}$ allows cheating with probability $\frac{3}{4}$.

In the real world, if $\mathcal{A}$ sent a proper quad in the commit phase, the envelope returned in the open phase is a random envelope and its value, r, satisfies $r = d \oplus b$. Inspection of the simulation shows that the same holds in the ideal world. if $\mathcal{A}$ sent an improper quad in the commit phase (conditioned on Alice not aborting), the envelope is randomly selected from one of the three containing the same bit, and its value satisfies $(1 - r) = d \oplus b$. Again, this holds in the ideal world.

Thus, $\mathcal{Z}$'s views are identically distributed in both worlds. □

Together, Lemmas 9.1 and 9.2 imply Theorem 4.2.

9.3. Amplification for weak bit commitment

The following protocol constructs an p^k-WBC using k instances of $\mathcal{F}^{(p-WBC)}$:

Commit b Alice chooses k random values $r_1, \ldots, r_k$ such that $r_1 \oplus \cdots \oplus r_k = b$. She commits to the values in parallel using $\mathcal{F}_1^{(p-WBC)}, \ldots, \mathcal{F}_1^{(k-WBC)}$.
Open Alice opens all k commitments. The result is the xor of the values returned.

The proof that this protocol securely realizes $\mathcal{F}^{(p^k-WBC)}$ is extremely similar to the proof of the RIS amplification protocol (in Section 8.2), and we omit it here. Letting $k = O(\log \frac{1}{\epsilon})$, the amplification protocol gives us the proof for Theorem 2.1.

10. Proof of security for oblivious transfer protocol

This section contains the proof of Theorem 5.1. The proof follows the standard scheme for proofs in the UC model (elaborated in Section 2.5). We deal separately with the case where $\mathcal{A}$ corrupts the sender and where $\mathcal{A}$ corrupts the receiver. Note that when $\mathcal{A}$ corrupts the sender, $\mathcal{I}$ simulates an honest receiver and references to steps in the protocol refer to Protocol 5.1b, while when $\mathcal{A}$ corrupts the receiver, $\mathcal{I}$ is simulating an honest sender and the steps refer to Protocol 5.1a.

10.1. $\mathcal{A}$ corrupts the receiver

Assume $\mathcal{A}$ begins by corrupting the receiver. $\mathcal{I}$ also corrupts the receiver and sends a **CanCheat** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$. $\mathcal{I}$ waits for the sender to send a **Send** command, then begins simulating the real-world protocol by sending nine **Receipt** messages to $\mathcal{A}$ (acting for the receiver). Call a triplet of containers in which all containers are sealed and all belonged to the original triplet *good*. We now describe a decision tree for $\mathcal{I}$. The edges in the tree correspond either to choices made by $\mathcal{A}$ (these are marked by †), or to responses from $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$.

Case $1^\dagger$: All the triplets created by the sender are returned by $\mathcal{A}$ at step (3) and all are good. In this case, $\mathcal{I}$ randomly chooses i as specified in the protocol and continues the simulation until step (5). The protocol continues depending on $\mathcal{A}$'s actions:
Case $1.1^\dagger$: $\mathcal{A}$ sent incorrect locations for the sender's triplets $\sigma(\{j_1, j_2\}) \neq \{1, 2, 3\} \setminus \{i\}$. In this case, the real sender would have aborted, so $\mathcal{I}$ aborts.

Case 1.2†: $\mathcal{A}$ sent correct locations for the triplets $\{1, 2, 3\} \setminus \{i\}$, but one of the triplets he wants the sender to return (k_1 or k_2) is actually triplet i. $\mathcal{I}$ chooses π_1 randomly as required by the protocol (note: below, we always refer to the permutation used shuffle the receiver's triplet as π_1 and the permutation used to shuffle the sender's triplet as π_2). The simulation continues depending on whether $\mathcal{I}$ cheated successfully:

Case 1.2.1: $\mathcal{I}$ cheated successfully and received b_0 and b_1 (this occurs with probability $\frac{1}{3}$). In this case, $\mathcal{I}$ continues the simulation until step (10), where $\mathcal{A}$ sends ℓ_2, its guess for $\pi_2(2)$, to the sender. At this point $\mathcal{I}$ always accepts (equivalently, it selects π_2 at random from the set of permutations for which $\pi_2(2) = \ell_2$). $\mathcal{I}$ can now continue simulating a real sender, following the protocol exactly.

Case 1.2.2: $\mathcal{I}$ failed to cheat and did not receive b_0, b_1. $\mathcal{I}$ continues the simulation until the end of step (10), where $\mathcal{A}$ sends ℓ_2, its guess for $\pi_2(2)$. At this point $\mathcal{I}$ always aborts (equivalently, it selects π_2 at random from the set of permutations for which $\pi_2(2) \neq \ell_2$, and continues the simulation for the sender, who will then abort).

Case 1.3†: $\mathcal{A}$ sent correct locations for the triplets $\{1, 2, 3\} \setminus \{i\}$ and both the triplets he asks the sender to return are the receiver's. In this case, $\mathcal{A}$ simulates the sender returning the two triplets to the receiver. $\mathcal{I}$ chooses a random bit a'. If the receiver asks to open his triplet, $\mathcal{I}$ returns answers consistent with the sender encoding a' on the receiver's triplet. $\mathcal{I}$ continues the simulation until step (24), when the receiver sends the bit b'. Since $\mathcal{I}$ knows σ, given b' $\mathcal{I}$ can compute the unique value, b, that is consistent with the input of an honest receiver using the same permutation σ and the same public messages. $\mathcal{I}$ sends a **Choice** b command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ and receives a_b. $\mathcal{I}$ then simulates the sender responding with $a_b \oplus a'$ to the receiver in stage (30). The simulation then continues until it $\mathcal{A}$ halts.

Case 2†: Of the triplets created by the sender, at most two are good and returned by $\mathcal{A}$ at step (3). Let j be the index of a bad (or missing) triplet (if there is more than one $\mathcal{I}$ chooses randomly between them). The simulation continues depending on whether $\mathcal{I}$ can cheat successfully:

Case 2.1: $\mathcal{I}$ received both b_0 and b_1 (this occurs with probability $\frac{1}{3}$). In this case, $\mathcal{I}$ chooses $i = j$. $\mathcal{I}$ then continues the protocol simulating an honest sender and letting the ideal receiver output whatever the $\mathcal{A}$ commands it to output.

Case 2.2: $\mathcal{I}$ cannot cheat successfully. In this case, $\mathcal{I}$ chooses i randomly from $\{1, 2, 3\} \setminus \{j\}$. This forces $\mathcal{A}$ to send the simulated sender the location of triplet j at step (5). No matter what he sends the real sender running the protocol in the "real-world" scenario would abort. Hence, $\mathcal{I}$ always aborts at step (6).

Lemma 10.1. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only the receiver, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. The proof is by case analysis. $\mathcal{I}$'s decision tree implicitly groups all possible adversaries by their actions at critical points in the protocol. To show that $\mathcal{Z}$'s view of the protocol is identically distributed in the real and ideal worlds, it is enough to show that the distribution of the view is identical given any specific choice by $\mathcal{Z}$ and $\mathcal{A}$. Since $\mathcal{I}$'s actions are identical for all adversaries in the same group, it is enough to consider the groups implied by $\mathcal{I}$'s decision tree.

Case 1.1 This is the case where $\mathcal{A}$ returned triplets that were all good, but sent incorrect locations for the sender's triplets. $\mathcal{Z}$'s view in this case consists only of **Receipt** messages, the index i that is chosen at random both in the real world and in the ideal world, and the $\perp$ message sent by the sender.

Case 1.2 This is the case where $\mathcal{A}$ returned triplets that were all good, but asked for triplet i instead of his own triplets. $\mathcal{Z}$'s view up to step (10) consists of the **Receipt** messages, the index i, the permutation π_1. All these are chosen identically in both the real and ideal worlds. In the real world, with probability $\frac{1}{3}$ the sender would have chosen π_2 that is inconsistent with $\mathcal{A}$'s guess ℓ_2, in which case the protocol would halt with the sender outputting $\perp$. In the ideal world, $\mathcal{I}$ can cheat with probability $\frac{1}{3}$, so with the same probability the protocol halts and the sender outputs $\perp$. Conditioned on the protocol not halting, the view in both cases is also identically distributed, because in the ideal world $\mathcal{I}$ cheated successfully and can simulate the real sender exactly (since it now knows a_0 and a_1).

Case 1.3 This is the case where the adversary follows the protocol exactly (as far as messages sent to the sender and the $\mathcal{F}^{(IWL)}$ functionality). In this case, $\mathcal{I}$ also simulates an honest sender exactly until step (13). In the real and ideal worlds, the bit encoded on Bob's triplet (a') is uniformly random. The response sent in stage (30) is in both cases completely determined (in the same way) by a', the input bits a_0, a_1 and the receiver's actions.

Case 2 This is the case where the adversary opened (or replaced) containers before returning them in stage (3). The view up to this stage in both the real and ideal world consists of **Receipt** messages and the ids of the opened containers (the contents are always 1 bits). In both the real and ideal worlds, the index i sent by the sender is uniformly distributed in $\{1, 2, 3\}$ (in the ideal world this is because the probability that $\mathcal{I}$ cheats successfully is $\frac{1}{3}$, so that with probability $\frac{1}{3}$, i is set to some fixed j, and with probability $\frac{2}{3}$ it is set to one of the other values). Also, in both worlds, the probability that the sender picked an index which was opened (replaced) by $\mathcal{A}$ is determined by the number of containers that were opened (and is at least $\frac{1}{3}$). In either case, $\mathcal{I}$ can exactly simulate the sender, since if cheating was unsuccessful the protocol will necessarily halt before $\mathcal{I}$ needs to use the sender's inputs. Thus, in both cases the protocol will be identically distributed. □

10.2. $\mathcal{A}$ corrupts the sender

Assume $\mathcal{A}$ begins by corrupting the sender. $\mathcal{I}$ corrupts the real sender and sends a **CanCheat** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$. $\mathcal{I}$ then runs the simulation until the simulated sender sends nine containers. $\mathcal{I}$ simulates the receiver returning the containers to the sender (note that the steps in the protocol now refer to Protocol 5.1b). The simulation now depends on $\mathcal{A}$'s actions:

Case 1†: $\mathcal{A}$ asks to open one of the containers before sending i to the receiver in step (6). Denote the index of the container $\mathcal{A}$ opens by j. $\mathcal{I}$ continues the simulation based on the response to the **CanCheat** command:

Case 1.1: $\mathcal{I}$ can cheat. In this case, $\mathcal{I}$ pretends $\mathcal{A}$ opened one of the sender's containers (chosen randomly); $\mathcal{I}$ selects a random permutation $\sigma \in S_6$ from the set of permutations that map one of the sender's containers to index j. $\mathcal{I}$ then continues the simulation to the end, as if the receiver was honest and had shuffled the containers using σ. If the simulation reaches stage (12) without anyone aborting, $\mathcal{I}$ sends a **Send** 0, 0 command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ and waits for the real (ideal dummy) receiver to send the **Choice** command. $\mathcal{I}$ then continues the simulation using the real receiver's bit. After the sender sends the bit in step (22), $\mathcal{I}$ calculates the simulated receiver's output and sends a **Resend** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ using that bit.

Case 1.2: $\mathcal{I}$ cannot cheat. In this case, $\mathcal{I}$ selects a random permutation $\sigma \in S_6$ from the set of permutations that map one of the receiver's containers to index j. $\mathcal{I}$ then continues the simulation as if the receiver had shuffled the containers using σ. No matter what $\mathcal{A}$ does, $\mathcal{I}$ will then abort at step (9), (13) or (17), since that is what the real receiver would have done.

Case 2†: $\mathcal{A}$ does not open any container before sending i in stage (6). The simulation continues until stage (9) or until $\mathcal{A}$ asks to open a container that should not be opened according to the protocol:

Case 2.1†: $\mathcal{A}$ does not open any container (except those called for by the protocol, which will always be $\mathcal{A}$'s own containers) until the beginning of stage (9). Note that in this case w.l.o.g., $\mathcal{A}$ can wait to open containers until step (11). $\mathcal{I}$ continues the simulation, randomly choosing σ at stage (10). The simulation can now take the following paths:

Case 2.1.1†: $\mathcal{A}$ does not open any container until step (12). By this stage the sender no longer holds any containers, so $\mathcal{A}$ cannot open containers later either. $\mathcal{I}$ continues the simulation using 0 in place of the receiver's choice bit. Since $\mathcal{I}$ knows what exchanges $\mathcal{A}$ made on each of the triplets, at the end of the protocol it can recover both a_0 and a_1. It sends a **Send** a_0, a_1 command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$.

Case 2.1.2†: $\mathcal{A}$ opens one of the containers before step (12).

Case 2.1.2.1: $\mathcal{I}$ can cheat. In this case, $\mathcal{I}$ pretends the container $\mathcal{A}$ opens belongs to the sender's triplet. $\mathcal{I}$ sends a **Send** 0, 0 command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ and waits for the real receiver to send the **Choice** command. $\mathcal{I}$ then continues the simulation using the real receiver's bit. After the corrupt sender sends the bit in stage (22), $\mathcal{I}$ calculates the simulated receiver's output and sends a **Resend** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ using that bit.

Case 2.1.2.2: $\mathcal{I}$ cannot cheat. In this case, $\mathcal{I}$ pretends the container $\mathcal{A}$ opens belongs to the receiver's triplet. Whatever $\mathcal{A}$ does, $\mathcal{I}$ will then abort in step (13) or (17).

Case 2.2†: $\mathcal{A}$ asks to open a container not called for by the protocol before stage (9). Denote the index of this container by j. $\mathcal{I}$'s actions depend on whether it can cheat:

Case 2.2.1: $\mathcal{I}$ can cheat. In this case, $\mathcal{I}$ selects a random permutation $\sigma \in S_6$ from the set of permutations that map one of the sender's containers to index j. $\mathcal{I}$ then continues the simulation to the end as if the receiver was honest and had shuffled the containers using σ. If the simulation reaches step (11) without anyone aborting, $\mathcal{I}$ sends a **Send** 0, 0 message to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$ and waits for the real receiver to send a **Choice** message. $\mathcal{I}$ continues the simulation using the real receiver's bit. At the end of the simulation, $\mathcal{I}$ knows the simulated receiver's output and uses that in a **Resend** command to $\mathcal{F}^{(\frac{1}{2},\frac{1}{3}-PCWOT)}$.

Case 2.2.2: $\mathcal{I}$ cannot cheat. In this case, $\mathcal{I}$ selects a random permutation $\sigma \in S_6$ from the set of permutations that map one of the receiver's containers to index j. $\mathcal{I}$ then continues the simulation as if the receiver had shuffled the containers using σ. If an opened container is sent to the receiver in step (8), $\mathcal{I}$ will then abort at stage (9), since that is what the real receiver would have done. If the opened container is not sent to the receiver at step (8), $\mathcal{I}$ will abort at step (13) or (17).

Lemma 10.2. *For any environment machine $\mathcal{Z}$, and any real adversary $\mathcal{A}$ that corrupts only the sender, the output of $\mathcal{Z}$ when communicating with $\mathcal{A}$ in the real world is identically distributed to the output of $\mathcal{Z}$ when communicating with $\mathcal{I}$ in the ideal world.*

Proof. The proof is by case analysis. $\mathcal{I}$'s decision tree implicitly groups all possible adversaries by their actions at critical points in the protocol. To show that $\mathcal{Z}$'s view of the protocol is identically distributed in the real and ideal worlds, it is enough to show that the distribution of the view is identical given any specific choice by $\mathcal{Z}$ and $\mathcal{A}$. Since $\mathcal{I}$'s actions are identical for all adversaries in the same group, it is enough to consider the groups implied by $\mathcal{I}$'s decision tree.

Case 1 This is the case where $\mathcal{A}$ first deviates from the protocol by opening one of the containers before sending i in step (6). In the real world, the receiver's choice of σ is uniformly random. Thus, no matter what container $\mathcal{A}$ chooses

to open, it will be one of the receiver's containers with probability $\frac{1}{2}$. In the ideal world, $\mathcal{I}$'s choice of σ is also random: with probability $\frac{1}{2}$, $\mathcal{I}$ can cheat, in which case σ is chosen from the half of S_6 permutations that map j to the sender's containers. With probability $\frac{1}{2}$, $\mathcal{I}$ cannot cheat, in which case σ is chosen from the half of S_6 permutations that map j to the receiver's containers. The rest of the simulation in the ideal world is an exact simulation of the real receiver (in the case that $\mathcal{I}$ cannot cheat, it will never need to use the sender's input bits, since it will halt in step (9), (13) or (17). Thus in both cases the protocol view is identically distributed.

Case 2.1.1 This is the case where $\mathcal{A}$ honestly follows the protocol (from the point of view of $\mathcal{I}$). In this case, up to stage (12), $\mathcal{I}$ simulates a real receiver exactly. The only difference between the simulation and the real world is that $\mathcal{I}$ uses the choice bit 0 in the simulation rather than the receiver's input bit. However, the view of $\mathcal{A}$ is identical, since in both cases the bit requested by the receiver in stage (12) is uniformly random (because σ is chosen at random, and $\mathcal{A}$ has no information about the order of the final two triplets). The receiver's output is identical in both worlds, because $\mathcal{I}$ can compute the sender's inputs from $\mathcal{A}$'s actions.

Case 2.1.2 This is the case where $\mathcal{A}$ first deviates from the protocol by opening a container during step (11). Up to the deviation from the protocol, $\mathcal{I}$ simulates the real receiver exactly, so the protocol view up to that point is identical in both worlds. In both worlds, $\mathcal{A}$ has no information about the order of the two remaining triplets (this is determined by the choice of σ and i). In the real world, the container $\mathcal{A}$ opens will be the receiver's container with probability $\frac{1}{2}$. In the ideal world, this will also be the case, since $\mathcal{I}$ can cheat with probability $\frac{1}{2}$. If $\mathcal{I}$ can cheat, the rest of the simulation exactly follows the protocol (since $\mathcal{I}$ now knows the real receiver's choice bit). If $\mathcal{I}$ cannot cheat, the choice of σ ensures that the rest of the simulation still follows the protocol exactly, since the receiver will abort before it needs to use its choice bit. Thus, in both worlds the protocol view is identically distributed.

Case 2.2 This is the case where $\mathcal{A}$ first deviates from the protocol by opening a container after sending i in step (6) but before stage (9). As in Case 1 (and for the same reasons), σ is uniformly distributed in both worlds. If $\mathcal{I}$ can cheat, the simulation follows the protocol exactly ($\mathcal{I}$ knows the real receiver's choice), so the view is identical. If $\mathcal{I}$ cannot cheat the choice of σ ensures that $\mathcal{I}$ will never have to use the real receiver's choice, so the view is again distributed identically to the real world. □

Together, Lemmas 10.1 and 10.2 prove Theorem 5.1.

11. Discussion and open problems

11.1. Zero knowledge without bit commitment

In the bare model, where BC is impossible, Zero-knowledge proofs exist only for languages in SZK — which is known to be closed under complement and is thus unlikely contain NP. An interesting open question is whether the class of languages that have zero-knowledge proofs in the DWL model (where BC is impossible; see Section 3.3) is strictly greater than SZK (assuming $P \neq NP$).

11.2. Actual human feasibility

The protocols we describe in this article can be performed by unaided humans, however, they require too many containers to be practical for most uses. It would be useful to construct protocols that can be performed with a smaller number of containers (while retaining security), and with a smaller number of rounds.

Another point worth mentioning is that the protocols we construct in the distinguishable models only require one of the parties to seal and verify containers. Thus, the binding property is only used in one direction, and the tamper-evidence and hiding properties in the other. This property is useful when we want to implement the protocols in a setting where one of the parties may be powerful enough to open the seal undetectably. This may occur, for instance, in the context of voting, where one of the parties could be "the government" while the other is a private citizen.

In both the weakly and strongly fair CF protocols, only the first round requires envelopes to be created, and their contents do not depend on communication with the other party. This allows the protocols to be implemented using scratch-off cards (which must be printed in advance). In particular, the weakly fair CF protocol can be implemented with a scratch-off card using only a small number of areas to be scratched.

In the case of BC, our protocol requires the powerful party to be the receiver. It would be interesting to construct a BC protocol for which the powerful party is the sender (i.e., only the sender is required to to seal and verify envelopes).

Acknowledgement

The first author's research was supported in part by a grant from the Minerva Foundation.

References

[1] Dorit Aharonov, Amnon Ta-Shma, Umesh V. Vazirani, Andrew C. Yao, Quantum bit escrow, in: STOC'00, 2000, pp. 705–714.
[2] Andris Ambainis, Markus Jakobsson, Helger Lipmaa, Cryptographic randomized response techniques, in: PKC'04, in: LNCS, vol. 2947, 2004, pp. 425–438.

[3] Ross J. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, John Wiley & Sons, Inc., 2001.
[4] Matt Blaze, Cryptology and physical security: Rights amplification in master-keyed mechanical locks, IEEE Security and Privacy (2003).
[5] Matt Blaze, Safecracking for the computer scientist. U. Penn CIS Department Technical Report, December 2004. http://www.crypto.com/papers/safelocks.pdf.
[6] Manuel Blum, Coin flipping over the telephone, in: Proceedings of IEEE COMPCON '82, 1982, pp. 133–137.
[7] Ran Canetti, Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000.
[8] Richard Cleve, Limits on the security of coin flips when half the processors are faulty, in: STOC '86, 1986, pp. 364–369.
[9] Claude Crépeau, Efficient cryptographic protocols based on noisy channels, in: Eurocrypt'97, in: LNCS, vol. 1233, 1997, pp. 306–317.
[10] Claude Crépeau, Joe Kilian, Achieving oblivious transfer using weakened security assumptions, in: FOCS'88, 1988, pp. 42–52.
[11] Ivan B. Damgård, Serge Fehr, Kiril Morozov, Louis Salvail, Unfair noisy channels and oblivious transfer, in: TCC'04, in: LNCS, vol. 2951, 2004, pp. 355–373.
[12] Ivan B. Damgård, Joe Kilian, Louis Salvail, On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions, in: Eurocrypt'99, in: LNCS, vol. 1592, 1999, pp. 56–73.
[13] Ronald Fagin, Moni Naor, Peter Winkler, Comparing information without leaking it, Commun. ACM 39 (5) (1996) 77–85.
[14] Sarah Flannery, David Flannery, In Code: A Mathematical Journey, Algonquin Books of Chapel Hill, 2002.
[15] Oded Goldreich, Silvio Micali, Avi Wigderson, Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems, J. ACM 38 (3) (1991) 691–729.
[16] Oded Goldreich, Ronen Vainish, How to solve any protocol problem — an efficiency improvement, in: CRYPTO'86, in: LNCS, vol. 263, 1987, pp. 73–86.
[17] Russell Impagliazzo, Michael Luby, One-way functions are essential for complexity based cryptography, in: FOCS'89, 1989, pp. 230–235.
[18] Joe Kilian, Founding crytpography on oblivious transfer, in: STOC'88, 1988, pp. 20–31.
[19] Hoi-Kwong Lo, H.F. Chau, Why quantum bit commitment and ideal quantum coin tossing are impossible, in: PhysComp'98, 1998, pp. 177–187.
[20] Dominic Mayers, Unconditionally secure quantum bit commitment is impossible, Phys. Rev. Lett. (78) (1997) 3414–3417.
[21] Tal Moran, Moni Naor, Polling with physical envelopes: A rigorous analysis of a human-centric protocol, in: Serge Vaudenay (Ed.), EUROCRYPT 2006, 2006, pp. 88–108. http://www.wisdom.weizmann.ac.il/~naor/PAPERS/polling.pdf.
[22] Tal Moran, Moni Naor, Gil Segev, An optimally fair coin toss: Cleve's bound is tight, in: TCC '09, 2009.
[23] Moni Naor, Yael Naor, Omer Reingold, Applied kid cryptography or how to convince your children you are not cheating, March 1999. http://www.wisdom.weizmann.ac.il/~naor/PAPERS/waldo.ps.
[24] Bruce Schneier, The solitaire encryption algorithm, 1999. http://www.schneier.com/solitaire.html.